REFINEKV: QUERY-ADAPTIVE KV CACHE COMPRESSION VIA ATTENTION-ERROR BOUNDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Repeatedly accessing the full key–value (KV) cache is a major bottleneck in long-context language model decoding. We find that small query-specific subsets can accurately approximate full attention, yet their union covers nearly the entire cache, motivating query-adaptive access for the KV cache. A natural approach is to split the KV cache into clusters, using compact representatives for clusters that incur negligible error while expanding the rest to their original KV entries. We derive a principled selection criterion, an upper bound on attention-output error, which reveals two complementary factors governing the error: *importance* measures the impact of a cluster, while *distortion* measures how faithfully the cluster can be represented. Based on this criterion, we propose REFINEKV, a decoding-time query-adaptive KV compression method. Experiments on two models and three benchmarks demonstrate that REFINEKV retains near-full-attention quality while achieving up to $11.7\times$ speedup in the attention phase and $1.76\times$ faster decoding, establishing a new quality–latency Pareto frontier.

1 INTRODUCTION

Context has emerged as a fundamental resource for modern language model inference in both reasoning LLMs (Guo et al., 2025) and interactive agent systems (Yao et al., 2023). During autoregressive decoding with full attention, each token computes attention weights between its query vector and the key vectors of preceding context tokens, then uses these weights to aggregate their value vectors (Vaswani et al., 2023). To avoid redundant computation, the key and value vectors for context tokens are stored as the Key-Value (KV) cache (Li et al., 2024). However, repeatedly accessing this massive KV cache at every generation step incurs severe overhead, making attention a substantial decoding bottleneck in long-context regimes (Tang et al., 2024).

KV cache compression aims to reduce the amount of data accessed by every decoding query while minimizing the deviation of compressed output from full attention output, typically through eviction (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2024; Oren et al., 2024), merging (Wang et al., 2024; Zhang et al., 2024) or quantization (Zirui Liu et al., 2023). To examine the potential for reducing KV access, we ask a key question: how much of the KV cache is necessary for each query to approximate attention output with high fidelity? Our findings reveal that a relatively small query-specific subset is already sufficient. However, these subsets vary substantially across queries, and their union covers nearly the entire KV cache. This gap reveals that different queries attend to sparse but different parts of the KV cache, which suggests that, rather than compressing the KV cache into a fixed subset for all future queries, the accessed subset should adapt to the current query (Tang et al., 2024; Hooper et al., 2025; Lu et al., 2025; Ni et al., 2026).

Query-adaptive access raises a central challenge: given a query, how can we decide which part should be accessed without reading the full KV cache? We move access decisions to a coarser granularity, using a compact summary to assess a group of KV entries without inspecting its individual members. Specifically, we partition the cache into clusters and precompute a compact summary for each cluster to guide query-dependent access. Selected clusters are expanded to expose their original KV entries, while the remaining clusters are approximated by compact representatives. Within this paradigm, the next question is: how can we identify clusters whose representative approximation introduces negligible attention-output error?

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

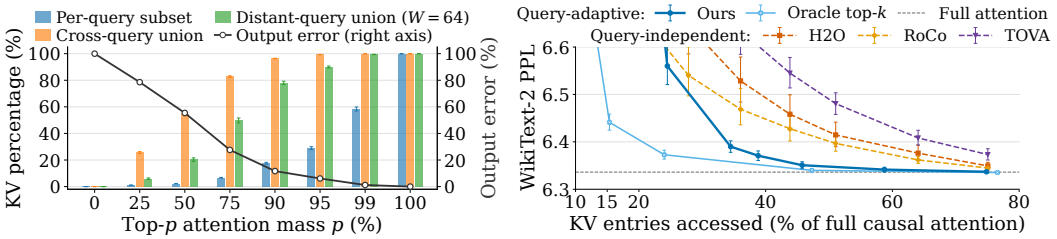


Figure 1: **Left: Each query needs only a small KV subset, but together these subsets cover nearly the entire cache.** For each query, we select the highest-weight KV entries needed to retain $p\%$ of its total attention weight. The figure compares sizes of these per-query subsets and their union across queries, alongside the resulting attention output error. At roughly 10% output error, each query needs only a small subset of the cache, yet their union covers most KV entries. **Right: Query-adaptive methods surpass query-independent methods.** Oracle query-adaptive top-k attention achieves a markedly better perplexity–access trade-off than representative eviction baselines. Our method recovers a substantial portion of this advantage without oracle access.

We address this question with a principled cluster-wise risk score. Starting from a decomposition of the attention-output error, we derive an upper bound on the overall approximation error and use its cluster-wise terms as risk scores. The cluster-wise score captures the joint effect of *global importance*, measured by a cluster's approximate attention mass, and *local distortion*, characterized by the spread of its attention logits under the current query. Under this criterion, high-importance clusters require tighter local approximation to remain within the same risk tolerance, while lower-importance clusters can tolerate greater distortion. Empirically, the resulting score correlates strongly with the actual attention-output error caused by cluster approximation.

Based on this risk score, we propose REFINEKV, a query-adaptive KV compression method for efficient decoding. After prefill, REFINEKV partitions the KV cache into contiguous clusters and precomputes their representatives and auxiliary metadata for computing risk score, while retaining the original KV entries for selective expansion. At each decoding step, cluster risk scores are computed from the current query and the cached summaries. Clusters whose risk score exceeds a predefined threshold are expanded to their original KV entries, while the remaining clusters are each compressed into a single representative. An attention operation over this mixed representation produces the approximate output. Experiments on two models and three benchmarks demonstrate that REFINEKV retains near-full-attention quality while achieving up to $11.7\times$ speedup in the attention phase and $1.76\times$ faster decoding, establishing a new quality–latency Pareto frontier.

2 MOTIVATION: FROM EVICTION TO QUERY-ADAPTIVE ACCESS

Autoregressive Transformers generate each token by conditioning on all previous tokens. Within an attention head, a query vector $\boldsymbol{q} \in \mathbb{R}^d$ representing the current decoding position interacts with the key vectors $\boldsymbol{k}_i \in \mathbb{R}^d$ of the context tokens at position i to determine how their value vectors $\boldsymbol{v}_i \in \mathbb{R}^{d_v}$ are aggregated. For a context of N tokens and query $\boldsymbol{q}$, the full-attention output $\boldsymbol{o} \in \mathbb{R}^{d_v}$:

$$\boldsymbol{o} = \sum_{i=1}^{N} \alpha_i \boldsymbol{v}_i, \qquad \alpha_i = \frac{\exp\left(\boldsymbol{q}^\top \boldsymbol{k}_i / \sqrt{d}\right)}{\sum_{j=1}^{N} \exp\left(\boldsymbol{q}^\top \boldsymbol{k}_j / \sqrt{d}\right)}, \tag{1}$$

During decoding, the keys and values of previous tokens are reused across generation steps. The Key-Value cache (KV cache) stores these representations as $\{(\boldsymbol{k}_i, \boldsymbol{v}_i)\}_{i=1}^{N}$, avoiding repeated computation. However, the cache grows linearly with context length N, and full attention reads the entire cache for every query, which still incurs substantial decoding latency in long-context regimes. To mitigate this, KV cache compression methods (Li et al., 2025; Zhang et al., 2023; Xiao et al., 2024) aim to reduce KV data accessed by each query while minimizing the deviation of the compressed output $\hat{\boldsymbol{o}}$ from full attention output $\boldsymbol{o}$ in Equation 1.

A fundamental question is: *For any query, how much of the KV cache needs to be accessed to accurately approximate full attention?* To examine this, we select the highest-weight KV entries

for each query until their cumulative attention mass reaches $p\%$. Figure 1 (left) shows how the percentage of KVs selected (bars) and the attention output error (line) vary with the amount of attention mass retained. The **blue bars** show the average fraction of KV items retained for each query. At an attention mass of $p = 90\%$, retaining merely 17.6% of the KV cache per query is sufficient to keep the approximation error down to 10%. This suggests that only a relatively small subset is crucial for accurately approximating the output for a particular query.

However, compressing the cache into a fixed subset requires that the same subset remains sufficient to accurately approximate the full-attention output across different queries. This raises a second question: *Are these crucial subsets shared across queries, such that a fixed subset of KV pairs could serve all future queries?* The **orange bars** show the size of union of these query-specific subsets. Although an individual query requires only 17.6% of the cache at $p = 90\%$, their union spans 96.4% of the KV cache. One potential confounding factor is attention locality: each query tends to attend strongly to its immediately preceding tokens. Thus, the union may appear large even under a purely local attention pattern. To control for this effect, we remove the most recent 64 tokens for each query before computing the union. The gap remains substantial, as shown by the **green bars**. This gap shows that different queries rely on different KV pairs, therefore a fixed subset of cache simply cannot work for all queries. Detailed experimental setups are provided in Appendix C.1.

These observations suggest a rethinking in the compression paradigm: **rather than committing to a fixed subset of KV entries for all future queries, the accessed subset should adapt to the current query.** We next examine whether such query-adaptive access indeed translates into a better end-to-end quality–efficiency trade-off. Figure 1 (right) illustrates a perplexity evaluation on WikiText-2. We consider three query-independent methods, H2O (Zhang et al., 2023), RoCo (Ren & Zhu, 2024), and TOVA (Oren et al., 2024), and contrast them with query-adaptive approaches: oracle top-k selection, and our practical query-adaptive approach introduced in the following section. The query-independent methods exhibit a substantially less favorable quality–budget trade-off. This provides end-to-end evidence supporting query-adaptive access as a more effective compression paradigm.

3 Deciding What to Access: From Attention Error to Risk Score

Query-adaptive access leaves a central question: which KV entries should be accessed for the current query? Answering this question involves two requirements: efficiency and accuracy. *Efficiency* requires making access decisions without scanning every token-level KV entry, which would otherwise defeat the purpose of compression. *Accuracy* requires selecting tokens that incur minimal output deviation from full attention.

3.1 Cluster-based Access

For *efficiency*, as KV entries cannot be visited individually, we need to move access decisions to a coarser granularity, using a compact summary to assess a group of KV entries, to decide whether they should be accessed. A natural realization of this idea is to partition KV entries into clusters, and use only cluster-level summaries to decide which clusters require token-level access. Selected clusters are expanded to compute full attention, while unselected ones are either pruned entirely (Tang et al., 2024; Lu et al., 2025) or coarsely approximated by their representative tokens (Hooper et al., 2025; Ni et al., 2026).

In this work, we adopt the representative-based approximation route: clusters whose coarse-grained representation introduces negligible error in the attention output are represented by their compact representatives, whereas those that cannot be faithfully approximated are expanded to full attention. Intuitively, this paradigm allocates a "high-resolution" view to clusters that require precise access. Then we focus on the *accuracy* requirement, which is: *how can we identify clusters whose approximation incurs negligible error in the attention output?*

3.2 Upper Bounding Attention Error

To answer this question, we seek an efficiently computable, query-dependent risk score that can reflect the output error caused by approximating a particular cluster. Specifically, we start from decomposing the single-head attention-output error. Let $\mathcal{C} = \{C_1, ..., C_m\}$ be a partition of the KV

cache. For each cluster C, we use its mean key and value $(\bar{\boldsymbol{k}}_C, \bar{\boldsymbol{v}}_C)$ as a shared representative for every KV entry in C. For a query $\boldsymbol{q}$, under full attention, let p_C denote the total attention weight assigned to cluster C, and let $\boldsymbol{v}_C$ denote the attention-weighted average of its token values. Under approximation, the cluster receives total attention weight $\widehat{p}_C$ and contributes its mean value $\bar{\boldsymbol{v}}_C$. Formally,

$$p_C = \frac{\sum_{i\in C} e^{\boldsymbol{q}^\top \boldsymbol{k}_i/\sqrt{d}}}{\sum_{j=1}^{N} e^{\boldsymbol{q}^\top \boldsymbol{k}_j/\sqrt{d}}}, \qquad \boldsymbol{v}_C = \frac{\sum_{i\in C} e^{\boldsymbol{q}^\top \boldsymbol{k}_i/\sqrt{d}} \boldsymbol{v}_i}{\sum_{j\in C} e^{\boldsymbol{q}^\top \boldsymbol{k}_j/\sqrt{d}}}, \qquad \widehat{p}_C = \frac{|C| e^{\boldsymbol{q}^\top \bar{\boldsymbol{k}}_C/\sqrt{d}}}{\sum_{C'\in\mathcal{C}} |C'| e^{\boldsymbol{q}^\top \bar{\boldsymbol{k}}_{C'}/\sqrt{d}}}. \quad (2)$$

The error between exact attention output $\boldsymbol{o} = \sum_{C\in\mathcal{C}} p_C \boldsymbol{v}_C$ and approximate attention output $\widehat{\boldsymbol{o}} = \sum_{C\in\mathcal{C}} \widehat{p}_C \bar{\boldsymbol{v}}_C$ can be decomposed as:

$$\boldsymbol{o} - \widehat{\boldsymbol{o}} = \sum_{C\in\mathcal{C}} p_C \boldsymbol{v}_C - \sum_{C\in\mathcal{C}} \widehat{p}_C \bar{\boldsymbol{v}}_C = \underbrace{\sum_{C\in\mathcal{C}} (p_C - \widehat{p}_C)\boldsymbol{v}_C}_{\text{inter-cluster mass redistribution}} + \underbrace{\sum_{C\in\mathcal{C}} \widehat{p}_C (\boldsymbol{v}_C - \bar{\boldsymbol{v}}_C)}_{\text{intra-cluster value distortion}}. \quad (3)$$

This decomposition separates the output error into two sources: The first term captures the redistribution of attention mass across clusters, whereas the second term captures the error caused by approximating $\boldsymbol{v}_C$ with $\bar{\boldsymbol{v}}_C$. After regrouping the first term (Lemma A.1), both terms are related to the dispersion the logits $\boldsymbol{q}^\top \boldsymbol{k}_i$ within a cluster: when all $\boldsymbol{q}^\top \boldsymbol{k}_i$ are close to $\boldsymbol{q}^\top \bar{\boldsymbol{k}}_C$, the cluster is well-approximated and both terms get close to zero.

Neither error term is directly computable from compact cluster summaries. We therefore bound these two terms using the maximum logit deviation: $\delta_C = \max_{i\in C} |\boldsymbol{q}^\top (\boldsymbol{k}_i - \bar{\boldsymbol{k}}_C)/\sqrt{d}|$. However, computing δ_C exactly would require scanning all keys in the cluster. We therefore use an efficiently computable upper bound $\delta_C^{\text{bound}} \geq \delta_C$, constructed from compact cluster summaries (Details in Appendix A.7). Letting $V_C = \max_{i\in C} \|\boldsymbol{v}_i\|_2$ and $V_{\max} = \max_{C\in\mathcal{C}} V_C$, we have:

$$\|\boldsymbol{o} - \widehat{\boldsymbol{o}}\|_2 \leq 2 \sum_{C\in\mathcal{C}} \underbrace{\widehat{p}_C \left(V_{\max} \frac{\cosh(\delta_C^{\text{bound}}) - 1}{\sum_{C'\in\mathcal{C}} \widehat{p}_{C'} \cosh(\delta_{C'}^{\text{bound}})} + V_C \tanh\left(\frac{\delta_C^{\text{bound}}}{2}\right)\right)}_{\text{Risk Score } R_C \text{ for cluster } C}. \quad (4)$$

In Eq. 4, we write the resulting upper bound in a clusterwise form and use each clusterwise term as its risk score R_C. The detailed derivation is provided in Appendix A. In the following section, we provide an analysis of this risk score: what it actually measures and whether it faithfully reflects the output error.

3.3 Importance and Distortion

The risk score derived from Eq. 4 captures two query-dependent factors that influence the approximation error: global importance, measured by approximated cluster attention mass $\widehat{p}_C$, and local distortion, measured by δ_C and upper-bounded by δ_C^{bound}.

Global Importance $\widehat{p}_C$. Approximation error depends on how much the cluster contributes to the overall attention computation. If a cluster receives only a negligible fraction of the attention mass relative to the rest of the context, even a relatively coarse approximation may have little effect on the final attention output.

Local Distortion δ_C^{bound}. For a given query, a cluster is easy to approximate when its tokens behave similarly under the query. In particular, if all $\boldsymbol{q}^\top \boldsymbol{k}_i$ are close to $\boldsymbol{q}^\top \bar{\boldsymbol{k}}_C$, replacing them by the representative introduces little error. Importantly, this distortion is query-dependent: a cluster may be dispersed in the original key space while remaining tightly concentrated along the direction relevant to the current query.

Crucially, the score captures the joint effect of both factors. A highly important cluster can only tolerate a small local approximation error, whereas a low-importance cluster can tolerate substantially larger internal variation. Figure 2 (Left) illustrates a sample from real data: Each point represents a cluster positioned by Local Distortion δ_C^{bound} and Global Importance $\widehat{p}_C$. The background color indicates the Risk score, and the tilted decision boundary $R_C = \tau$ adaptively balances $\widehat{p}_C$ against δ_C^{bound}.

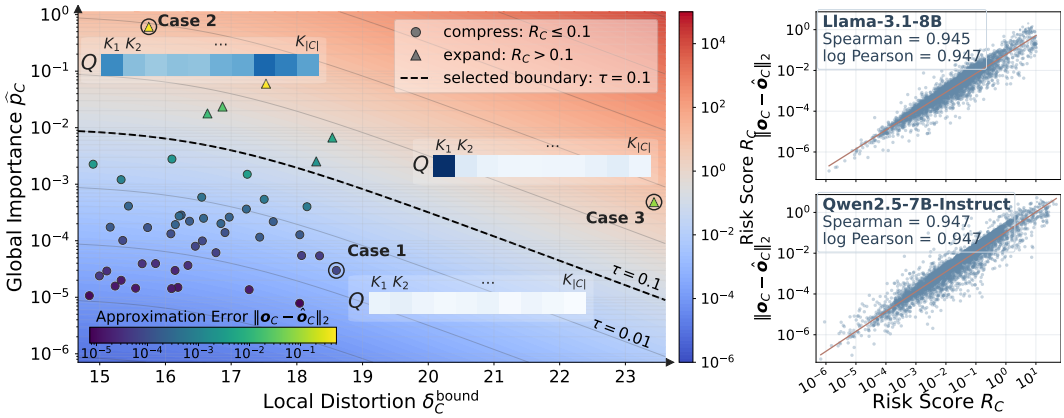


Figure 2: **Left: Cluster selection is jointly determined by importance and distortion.** Each point represents a cluster positioned by its global importance $\widehat{p}_C$ and local distortion δ_C^{bound}. Background color encodes the risk score R_C, with the dashed line denoting the decision boundary at $R_C = \tau$. The tilted boundary captures their trade-off: important clusters tolerate only small local distortion, whereas low-importance clusters can tolerate larger distortion. Point colors indicate the error incurred on approximating cluster. Three cases show the within-cluster attention-score patterns. **Right: Risk score R_C strongly correlated with approximation error.** These figures illustrate the correlation between Risk Score and cluster approximation error $\|\boldsymbol{o}_C - \hat{\boldsymbol{o}}_C\|_2$, where $\boldsymbol{o}_C$ and $\hat{\boldsymbol{o}}_C$ denote original output and approximate output for cluster C.

Beyond this theoretical intuition, the next questions is whether risk score R_C faithfull reflect the true output error caused by approximation. Figure 2 (Right) plots the risk score against the cluster approximation error $\|\boldsymbol{o}_C - \hat{\boldsymbol{o}}_C\|_2$, where $\boldsymbol{o}_C$ and $\hat{\boldsymbol{o}}_C$ denote original output and approximate output for cluster C. The results demonstrate a clear, robust correlation across different model architectures, confirming that our theoretical bound serves as an effective surrogate for actual approximation error.

4 REFINEKV: EFFICIENT RISK-GUIDED KV ACCESS

The preceding analysis provides a principled Risk Score for deciding which clusters require token-level access. Turning this criterion into an efficient decoding method requires both making access decisions without scanning the full KV cache and ensuring that these decisions reduce actual KV reads. We address these requirements with REFINEKV, a query-adaptive KV access method that preserves the full cache while selectively accessing its original entries.

Pipeline Overview. We illustrate our pipeline in Figure 3. In prefill phase, we first compute the entire KV cache and then partition it into contiguous clusters and precompute cluster representative mean key and value $(\bar{\boldsymbol{k}}_C, \bar{\boldsymbol{v}}_C)$ as well as auxiliary metadata. At each decoding step t, the incoming query $\boldsymbol{q_t}$ interacts only with these cluster-level summaries to estimate the approximation risk score R_C. Clusters with risk above a predefined threshold τ are expanded for token-level attention, while the remaining clusters are represented by their cached means. A single attention operation over this mixed representation produces the approximate output.

Efficient Estimation of Risk Score. Evaluating the risk score R_C in Eq. 4 requires four quantities: $\widehat{p}_C$, $V_{\max}$, V_C, and δ_C^{bound}. Among these, $\widehat{p}_C$ is computed directly from the cluster size $|C|$ and centroid $\bar{\boldsymbol{k}}_C$ via Eq. 2, while V_C and $V_{\max}$ are precomputed cluster-level statistics. Finally, δ_C^{bound} is derived from compact cluster summaries, with details deferred to Appendix A.7.

Determining the Expansion Set. Once the risk scores are available, we compare them with a threshold τ to determine which clusters should be expanded for token-level attention. For a single query head, the rule is straightforward: expand clusters with $R_C > \tau$ and approximate the remaining clusters by their representatives. Modern large language models adopt Grouped-Query Attention (GQA) (Ainslie et al., 2023), where multiple query heads share a single key-value head. Because selection is executed at the KV-cache level, all query heads sharing a KV group must operate on a unified set of expanded clusters. We designate **GQA-any** as our default strategy: a cluster C is

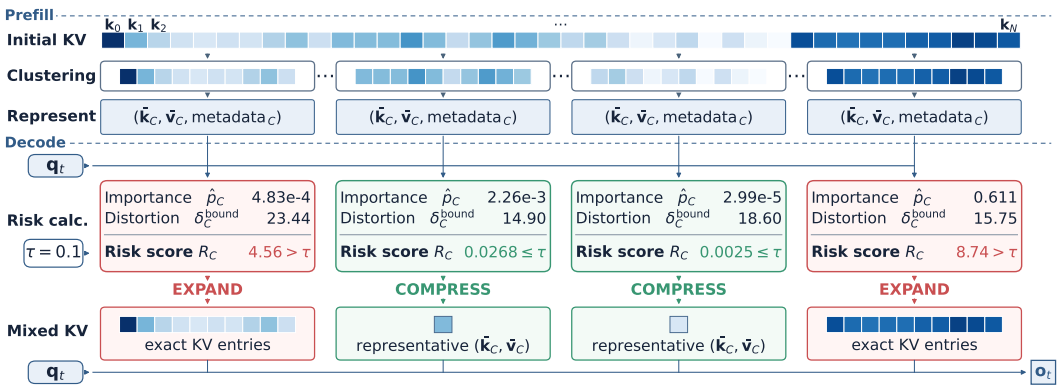


Figure 3: **REFINEKV Pipeline.** Prefill phase: The KV cache is partitioned into contiguous clusters, and a representative key-value pair together with auxiliary metadata is computed for each cluster. Decode phase: Each query $\boldsymbol{q}_t$ uses the cached summaries to calculate risk score R_C. Clusters with $R_C > \tau$ are expanded, while clusters with $R_C \leq \tau$ retain only their representatives. The resulting mixed KV sequence is used to compute the output $\mathbf{o}_t$. The color depth of each block indicates attention score.

expanded if the risk score evaluated by *any* query head within the group exceeds the threshold. We also consider an alternative aggregation heuristic, **GQA-mean**, which expands a cluster only when the average risk across the group exceeds τ; we systematically ablate both approaches in Section 5.2.

Efficient kernel implementation. Realizing the algorithmic sparsity of REFINEKV requires reducing actual KV-cache traffic. We first compute and cache the representative and bound statistics of each prompt block. At each decoding step, a lightweight block-selection kernel evaluates the expansion criterion of REFINEKV using the current query and these cached summaries, producing a mask that identifies the blocks requiring exact attention. The sparse-attention kernel consumes this mask directly on the GPU: it loads token-level K/V only from the selected blocks, while approximating the remaining blocks with their representatives. All contributions are incrementally combined using online softmax, without computing token-level scores for the entire prompt or gathering selected K/V into temporary buffers. Appendix B.2 provides the implementation details.

5 EXPERIMENTS

5.1 EXPERIMENT SETUP

Models. Experiment is done on two open-sourced long context models from different model families: Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-7B-Instruct (Qwen et al., 2025). Both model support up to 128K context, which is the default setting we use for evaluation.

Benchmarks. We evaluate long-context generation on three representative benchmarks: LongBench (Bai et al., 2024), RULER (Hsieh et al., 2024), and HELMET (Yen et al., 2025), which cover complementary application scenarios. LongBench measures real-world long-document understanding across question answering, summarization, retrieval, and code completion. RULER uses controlled synthetic tasks to evaluate retrieval, association, and aggregation as the context length increases. HELMET covers diverse application-oriented workloads, testing whether the quality–efficiency trade-off generalizes across different patterns of long-context usage.

Baselines. We compare REFINEKV with Full Attention and five representative efficient-attention baselines. SnapKV (Li et al., 2024) compresses the prompt KV cache by using attention statistics from a recent observation window. StreamingLLM (Xiao et al., 2024) preserves attention-sink tokens together with a sliding window. PyramidKV (Cai et al., 2025) adaptively distributes the KV budget across layers. Quest (Tang et al., 2024) partitions the KV cache into pages and uses query-dependent key bounds to select pages that may receive high attention. Double-P (Ni et al., 2026) clusters prompt KVs and applies two-stage probability-mass pruning.

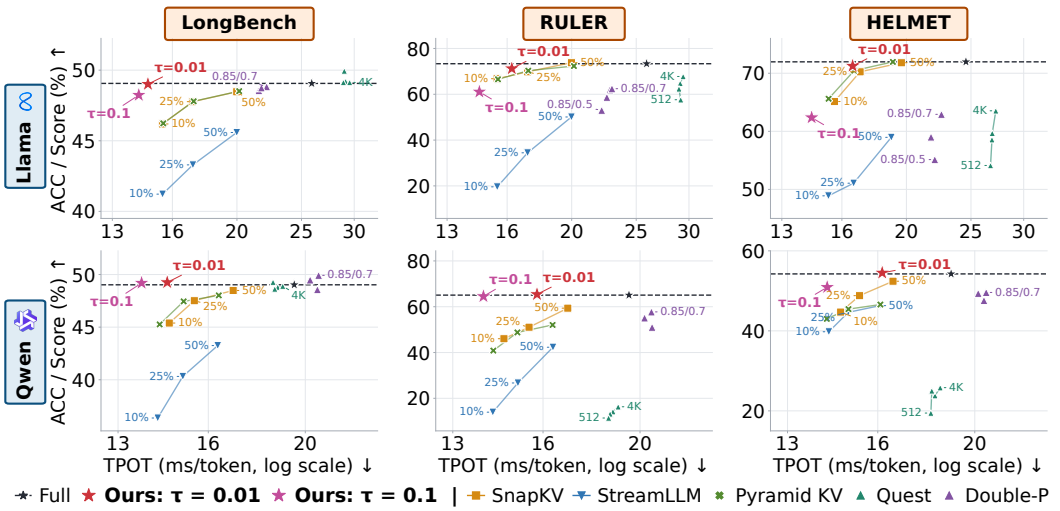


Figure 4: **Quality-latency trade-off of REFINEKV and existing KV-cache compression baselines.** Experiments are evaluated across two models (Llama-3.1 and Qwen-2.5) on three long-context benchmarks (LongBench, RULER, and HELMET), with results averaged across all tasks. We present a conservative threshold $\tau = 0.01$, which matches full attention performance across all settings while achieving up to $1.76\times$ decode speedup as measured by Time Per Output Token (TPOT). Our method consistently extends the quality–latency Pareto frontier.

Metric. We use measured latency, rather than the retained KV budget, as our primary efficiency metric because methods with similar token budgets can have substantially different runtimes. Since prefill is performed only once and contributes a relatively small fraction of the total cost (Analysis in Appendix C.2.4), we focus on decode time per output token (TPOT), which reflects end-to-end decode performance rather than isolated attention phase latency. For quality, we follow the official metric of each task and report the unweighted macro-average across tasks.

Ours. We partition the KV sequence into contiguous clusters of 32 tokens and report the quality–latency trade-off using two thresholds, $\tau \in \{0.01, 0.1\}$. For the RULER and HELMET benchmarks, we enable 32-query anchors: we keep expanding cluster selected by the last 32 prompt queries. For LongBench, we use no query anchors. We provide details of query anchor in Appendix B.1. For GQA aggregation, we use GQA-any introduced in Section 4.

Detailed model configurations, benchmark setups, baseline implementations, evaluation metrics, and hardware settings are provided in Appendix C.2.

5.2 MAIN RESULTS

Figure 4 shows that REFINEKV consistently provides a favorable quality–latency trade-off across both models and all three benchmarks. With the conservative threshold $\tau = 0.01$, REFINEKV nearly matches the quality of Full Attention in all six model–benchmark settings while reducing TPOT, achieving up to a $1.76\times$ decode speedup over Full Attention. Compared with both eviction-based and query-adaptive baselines, REFINEKV introduces new Pareto-optimal operating points, extending the empirical frontier toward lower latency without sacrificing quality. Increasing the threshold to $\tau = 0.1$ enables more aggressive sparsification and further improves the maximum speedup to $1.82\times$, although its effect on quality varies across tasks. Table 1 reports quality–latency results by task group, while Appendix C.3.1 provides the per-task results.

Outperform full attention in certain tasks. Interestingly, our method can outperform full attention on tasks dominated by irrelevant context like LongBench-GovReport or RULER-NIAH-MultiKey. One possible explanation is that preserving salient blocks exactly while representing less relevant blocks acts as an adaptive information filter, allowing the model to focus more on task-relevant evidence.

Table 1: Task-group quality–latency comparison. Each cell reports Accuracy/Acceleration, where Accuracy is the task-macro benchmark score and Acceleration is latency acceleration relative to Full, higher is better for both. Shaded cells are Pareto-optimal. Underlined cells have an absolute Accuracy drop versus Full of less than 0.5 point. Baselines use SnapKV/StreamLLM/PyramidKV.

	Method	LongBench					RULER			HELMET			
		Doc QA	Summ.	Few-shot	Synth.	Code	NIAH	A&T	QA	RAG	LC-QA	ICL	Rec./Rank.
Llama	Full	43.8/1.0	29.1/1.0	69.2/1.0	53.7/1.0	59.8/1.0	88.4/1.0	45.3/1.0	55.0/1.0	60.4/1.0	48.6/1.0	77.0/1.0	76.6/1.0
	Snap 10%	41.2/1.7	23.4/1.7	65.9/1.7	53.4/1.7	58.5/1.7	79.5/1.7	40.7/1.7	55.0/1.7	61.0/1.6	48.4/1.6	77.0/1.6	60.7/1.5
	Snap 25%	42.8/1.5	26.3/1.5	67.5/1.5	53.2/1.5	59.8/1.5	84.2/1.5	41.3/1.5	55.0/1.5	60.6/1.5	49.4/1.5	78.0/1.4	71.8/1.4
	Str. 10%	32.7/1.7	22.2/1.7	63.3/1.7	47.5/1.7	56.1/1.7	11.2/1.7	29.8/1.7	37.5/1.7	53.5/1.6	42.4/1.6	75.0/1.6	8.6/1.5
	Str. 25%	35.3/1.5	24.7/1.5	66.1/1.5	46.2/1.5	58.1/1.5	30.6/1.5	34.2/1.5	50.0/1.5	57.9/1.5	41.5/1.5	73.0/1.4	17.3/1.4
	Pyr. 10%	41.3/1.7	23.6/1.7	66.0/1.7	53.3/1.7	58.4/1.7	79.7/1.7	39.2/1.7	55.0/1.7	61.0/1.6	49.0/1.6	75.0/1.6	63.9/1.5
	Pyr. 25%	43.0/1.5	26.2/1.5	67.5/1.5	53.0/1.5	59.8/1.5	84.8/1.5	41.8/1.5	55.0/1.5	60.4/1.5	49.4/1.5	76.0/1.4	74.4/1.4
	Our τ.01	**43.6**/1.8	**29.2**/1.8	**69.5**/1.7	**53.4**/1.8	**59.9**/1.8	**84.5**/1.6	**46.6**/1.6	**55.0**/1.5	**60.4**/1.5	**48.9**/1.5	**76.0**/1.4	**76.3**/1.4
	Our τ.1	**43.3**/1.8	**28.4**/1.8	**67.6**/1.8	**52.4**/1.8	**59.4**/1.8	**70.5**/1.8	**40.0**/1.8	**55.0**/1.8	**58.5**/1.7	**47.9**/1.7	**79.0**/1.6	**54.2**/1.6
Qwen	Full	43.5/1.0	26.4/1.0	69.2/1.0	54.5/1.0	63.6/1.0	73.8/1.0	56.9/1.0	42.5/1.0	49.0/1.0	51.7/1.0	76.0/1.0	39.3/1.0
	Snap 10%	41.3/1.3	21.3/1.3	62.2/1.3	54.5/1.3	59.5/1.3	45.3/1.3	53.6/1.3	37.5/1.3	46.7/1.3	49.9/1.3	74.0/1.3	12.7/1.3
	Snap 25%	42.8/1.2	23.9/1.2	66.2/1.2	54.5/1.2	62.0/1.3	51.2/1.2	57.9/1.2	40.0/1.2	49.2/1.2	49.1/1.2	75.0/1.2	22.2/1.2
	Str. 10%	30.1/1.3	19.7/1.3	62.8/1.3	23.2/1.3	53.8/1.3	6.9/1.3	27.1/1.3	22.5/1.3	38.1/1.3	46.3/1.3	70.0/1.3	6.3/1.3
	Str. 25%	33.0/1.2	22.5/1.2	65.9/1.2	33.1/1.2	57.8/1.2	22.0/1.2	30.3/1.2	40.0/1.2	39.8/1.2	47.3/1.2	75.0/1.2	10.2/1.2
	Pyr. 10%	40.5/1.3	22.2/1.3	61.9/1.3	54.5/1.3	60.0/1.3	38.1/1.3	52.0/1.3	35.0/1.3	47.3/1.3	49.0/1.3	71.0/1.3	10.2/1.3
	Pyr. 25%	42.1/1.3	24.2/1.3	66.9/1.3	54.5/1.3	62.1/1.3	48.8/1.3	54.7/1.3	40.0/1.3	45.6/1.2	48.7/1.2	75.0/1.2	13.5/1.2
	Our τ.01	**43.0**/1.3	**27.7**/1.3	**69.8**/1.2	**54.5**/1.3	**64.2**/1.3	**74.4**/1.2	**56.7**/1.2	**42.5**/1.1	**48.5**/1.1	**50.8**/1.1	**76.0**/1.1	**40.5**/1.2
	Our τ.1	**43.0**/1.4	**28.0**/1.4	**69.9**/1.4	**54.0**/1.4	**63.6**/1.4	**72.2**/1.3	**57.1**/1.3	**45.0**/1.3	**48.5**/1.3	**51.0**/1.3	**76.0**/1.3	**34.2**/1.3

Latency Breakdown. Figure 5 shows that RefineKV substantially reduces decoding latency. Under BF16, it achieves an 11.22× attention speedup and a 1.73× overall decoding speedup. The overall gain is limited by non-attention operations, particularly the GEMV kernels. Under FP8, the overall decoding speedup increases to 2.13×. Compared with the ideal attention acceleration implied by the KV-access budget (shaded), we realize 97.0% of the acceleration headroom, demonstrating that our implementation operates close to the budget-constrained optimum.

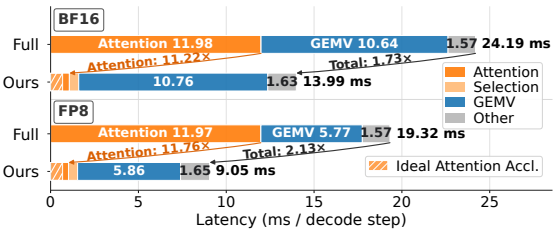


Figure 5: **Latency Breakdown.** Results are measured on Llama-3.1-8B-Instruct with 128K input.

Ablation Study. We conduct an ablation study to evaluate the effect of each design choice. Table 2 shows the ablation of the threshold τ, cluster size $|C|$, GQA reduction method. **Ours** is the default configuration, which is $\tau = 0.01, |C| = 32$, GQA-any. We report the accuracy score, TPOT and speedup relative to full attention. We provide ablation study for Q-anchor in Appendix B.1.

Table 2: Ablation study of Llama-3.1-8B-Instruct on the fixed RULER-128K evaluation subset.‘

Metric	Ref.		Threshold τ			$\lvert C\rvert$		GQA
	Full	**Ours**	0.001	0.1	1	16	64	Mean
Acc↑	73.35	**70.94**	73.49	61.05	28.74	71.42	72.58	69.65
TPOT↓	26.52	**16.35**	21.18	14.52	14.38	16.04	17.91	15.35
Speed↑	1.00×	**1.62**×	1.25×	1.83×	1.84×	1.65×	1.48×	1.73×

6 Related Works

Eviction-based KV Cache Compression. A prominent line of work reduces KV-cache overhead by permanently retaining only a subset of tokens deemed important (Zhang et al., 2023; Xiao et al., 2024; Cai et al., 2025; Guo et al., 2024; Oren et al., 2024; Ren & Zhu, 2024). H_2O (Zhang et al., 2023) identifies *heavy-hitter* tokens from accumulated attention statistics and dynamically preserves them together with recent tokens. StreamingLLM (Xiao et al., 2024) exploits the attention-sink phenomenon and maintains a small set of initial sink tokens alongside a sliding window of recent tokens. Another line of work compresses the cache by merging multiple KV states into fewer rep-

resentatives (Zhang et al., 2024; Wan et al., 2025; Wang et al., 2024; Yuan et al., 2025; Hu et al., 2026); while retaining more aggregate information than hard eviction, such merging is still generally lossy because the original token-level states cannot be exactly recovered once combined. Despite their different compression mechanisms, these methods make irreversible decisions about the KV cache: discarded or merged token-level states are no longer fully accessible to future queries.

Query-Aware Cluster-Based Attention. To avoid the irreversible information loss, another line of work preserves recoverable access to the original KV states and performs query-conditioned sparse attention at inference time. Block-based methods (Tang et al., 2024; Lu et al., 2025) partition the KV cache into contiguous pages and use compact page-level statistics to identify regions relevant to the current query . Quest (Tang et al., 2024) estimates the importance of each page using coordinate-wise key extrema and loads only the highest-scoring pages for exact attention. More recent methods, including Multipole Attention (Hooper et al., 2025), and Double-P (Ni et al., 2026), further adopt a coarse-to-fine, mixed-resolution design. These approaches establish block and cluster summaries as an efficient interface for query-dependent access to long contexts. However, their refinement decisions are primarily driven by estimated attention scores, rather than by a direct characterization of the attention-output error introduced when a cluster remains at low resolution.

Adaptive Sparse-Attention Selection. Recent KV compression and sparse attention methods have increasingly explored adaptive computation budgets to accommodate the heterogeneous attention patterns across heads, layers, and queries (Feng et al., 2025; Cai et al., 2025). A fixed top-k budget cannot adapt to the substantial variation in attention concentration across queries, heads, and layers. Twilight (Lin et al., 2025) addresses this limitation by introducing top-p pruning, which dynamically adjusts the retained set. Tactic (Zhu et al., 2025) uses function fitting to estimate the weight distributions under top-p clustering. Our method provides a complementary route to adaptive budgeting: instead of explicitly prescribing or allocating a budget, we expand clusters according to their query-conditioned approximation risk. Consequently, the KV access budget emerges naturally from error control, allocating more computation only when a coarse representation is insufficient to faithfully approximate the attention output.

7 Limitations and Future Work

Our work has several limitations that suggest promising directions for future research.

First, our theoretical analysis focuses on the approximation error of a single-head attention output and does not characterize how such errors propagate through Transformer layers. Extending the analysis toward end-to-end guarantees remains a challenging yet important direction.

Second, we currently use the average key and value as the representative of each cluster. While this choice is simple and efficient, more expressive representative-selection strategies may better preserve the information relevant to attention computation. Future work could explore weighted representatives or multiple representatives per cluster.

Finally, our current method adopts contiguous clustering. A natural extension is hierarchical clustering, where decoding starts from coarse cluster summaries and progressively expands only the regions that require higher-resolution access. Such a hierarchy could enable flexible allocation of KV access budgets and provide a better trade-off between accuracy and decoding efficiency.

8 Conclusion

In this work, we revisit KV compression from the perspective of query-adaptive access. Starting from attention output error, we derive a query-dependent cluster-wise risk score that characterizes whether each cluster can be faithfully approximated. Based on this criterion, we present RefineKV, a query-adaptive KV access method that preserves the full cache while selectively expanding clusters that require fine-grained attention. Experiments show that RefineKV achieves a favorable trade-off between quality and efficiency.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods and assisting with the design or feedback on research methodology or experiments. We have not used generative AI tools for generating synthetic data sets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, or interpreting results, and translation tasks requiring disclosure are not applicable to this work. Additionally, we used generative AI tools for editing the research paper to improve readability, creating or editing software code, and suggesting experimental parameters. We have reviewed all AI-assisted work. Specifically, all AI-suggested text edits were manually examined to preserve original technical claims, and all generated code, helper scripts, and experimental settings were rigorously reviewed, unit-tested, and verified against baselines by the authors to guarantee correctness and reproducibility. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding, 2024. URL https://arxiv.org/abs/2308.14508.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2025. URL https://arxiv.org/abs/2406.02069.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: Kv cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025. URL https://arxiv.org/abs/2510.00636.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference, 2025. URL https://arxiv.org/abs/2407.11550.

Aaron Grattafiori, Abhimanyu Dubey, and Abhinav Jauhri et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Daya Guo, Dejian Yang, Haowei Zhang, and et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters, 2024. URL https://arxiv.org/abs/2406.12335.

Coleman Hooper, Sebastian Zhao, Luca Manolache, Sehoon Kim, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Multipole attention for efficient long context reasoning, 2025. URL https://arxiv.org/abs/2506.13059.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Jie Hu, Shengnan Wang, Yutong He, Ping Gong, Jiawei Yi, Juncheng Zhang, Youhui Bai, Renhai Chen, Gong Zhang, Cheng Li, and Kun Yuan. Centroidkv: Efficient long-context llm inference via kv cache clustering, 2026. URL https://arxiv.org/abs/2506.11418.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on kv cache management, 2025. URL https://arxiv.org/abs/2412.19442.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation, 2024. URL https://arxiv.org/abs/2404.14469.

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning, 2025. URL https://arxiv.org/abs/2502.02770.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms, 2025. URL https://arxiv.org/abs/2502.13189.

Wentao Ni, Kangqi Zhang, Zhongming Yu, Oren Nelson, Mingu Lee, Hong Cai, Fatih Porikli, Jongryool Kim, Zhijian Liu, and Jishen Zhao. Double-p: Hierarchical top-p sparse attention for long-context llms, 2026. URL https://arxiv.org/abs/2602.05191.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns, 2024. URL https://arxiv.org/abs/2401.06104.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models, 2026. URL https://arxiv.org/abs/2309.00071.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Siyu Ren and Kenny Q. Zhu. On the efficacy of eviction policy for key-value constrained generative language model inference, 2024. URL https://arxiv.org/abs/2402.06262.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. D2o: Dynamic discriminative operations for efficient long-context inference of large language models, 2025. URL https://arxiv.org/abs/2406.13035.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks, 2024. URL https://arxiv.org/abs/2407.08454.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. Helmet: How to evaluate long-context language models effectively and thoroughly, 2025. URL https://arxiv.org/abs/2410.02694.

Jian Yuan, Ziwei He, Haoli Bai, Jingwen Leng, and Bo Jiang. Weightedkv: Attention scores weighted key-value cache merging for large language models, 2025. URL https://arxiv.org/abs/2503.01330.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. CaM: Cache merging for memory-efficient LLMs inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 58840–58850. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhang24n.html.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Kan Zhu, Tian Tang, Qinyu Xu, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context llms, 2025. URL https://arxiv.org/abs/2502.12216.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi : Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization. 2023. doi: 10.13140/RG.2.2.28167.37282. URL https://www.researchgate.net/doi/10.13140/RG.2.2.28167.37282.

A Derivation for cluster selection criterion

We provide a detailed derivation of our selection criterion in the following section. We present the formulation for a single attention head and omit layer and head indices for clarity.

A.1 Setup.

Let $\{(\boldsymbol{k}_i, \boldsymbol{v}_i)\}_{i=1}^N$ denote the KV cache, where $\boldsymbol{k}_i \in \mathbb{R}^d, \boldsymbol{v}_i \in \mathbb{R}^{d_v}$. Let $\mathcal{C} = \{C_1, \dots, C_m\}$ be a partition of the KV cache. For each cluster $C \in \mathcal{C}$, let $n_C = |C|$ and define

$$\bar{\boldsymbol{k}}_C = \frac{1}{n_C} \sum_{i \in C} \boldsymbol{k}_i, \qquad \bar{\boldsymbol{v}}_C = \frac{1}{n_C} \sum_{i \in C} \boldsymbol{v}_i. \tag{5}$$

For a query $\boldsymbol{q} \in \mathbb{R}^d$, define the token and representative logits as

$$s_i = \frac{\boldsymbol{q}^\top \boldsymbol{k}_i}{\sqrt{d}}, \qquad s_C = \frac{\boldsymbol{q}^\top \bar{\boldsymbol{k}}_C}{\sqrt{d}}. \tag{6}$$

The exact and approximated unnormalized masses of cluster C are

$$z_C = \sum_{i \in C} e^{s_i}, \qquad \widehat{z}_C = n_C e^{s_C}. \tag{7}$$

Equivalently, the representative of C has logit $s_C + \log n_C$.

The exact and approximated total mass are:

$$Z = \sum_{C \in \mathcal{C}} z_C, \qquad \widehat{Z} = \sum_{C \in \mathcal{C}} \widehat{z}_C. \tag{8}$$

Define the exact within-cluster output

$$\boldsymbol{v}_C = \frac{\sum_{i\in C} e^{s_i}\boldsymbol{v}_i}{z_C}, \tag{9}$$

and the exact and approximated cluster probabilities

$$p_C = \frac{z_C}{Z}, \qquad \widehat{p}_C = \frac{\widehat{z}_C}{\widehat{Z}}. \tag{10}$$

The exact and approximated attention outputs are therefore

$$\boldsymbol{o} = \sum_{C\in\mathcal{C}} p_C \boldsymbol{v}_C, \qquad \widehat{\boldsymbol{o}} = \sum_{C\in\mathcal{C}} \widehat{p}_C \bar{\boldsymbol{v}}_C. \tag{11}$$

Let

$$V_{\max} = \max_i \|\boldsymbol{v}_i\|_2, \qquad V_C = \max_{i\in C} \|\boldsymbol{v}_i\|_2. \tag{12}$$

A.2 Decomposition of attention output.

We first decompose the single-head attention output into **inter-cluster mass redistribution** and **intra-cluster value distortion**. Defining the auxiliary output $\widetilde{\boldsymbol{o}} = \sum_{C\in\mathcal{C}} \widehat{p}_C \boldsymbol{v}_C$, we have:

$$\boldsymbol{o} - \widehat{\boldsymbol{o}} = (\boldsymbol{o} - \widetilde{\boldsymbol{o}}) + (\widetilde{\boldsymbol{o}} - \widehat{\boldsymbol{o}}) \tag{13}$$

$$= \underbrace{\sum_{C\in\mathcal{C}} (p_C - \widehat{p}_C)\boldsymbol{v}_C}_{\text{inter-cluster mass redistribution}} + \underbrace{\sum_{C\in\mathcal{C}} \widehat{p}_C(\boldsymbol{v}_C - \bar{\boldsymbol{v}}_C)}_{\text{intra-cluster value distortion}}. \tag{14}$$

This decomposition separates the output error into two sources: inter-cluster mass redistribution and intra-cluster value distortion. It reveals that the approximation error arises through two mechanisms: The first term, $(p_C - \widehat{p}_C)$, reflects how the estimated cluster probability mass deviates from the real mass, capturing the redistribution of attention mass *across clusters* caused by approximating with cluster representatives.

The second term describes the error caused by approximating $\boldsymbol{v}_C$ with $\bar{\boldsymbol{v}}_C$. It satisfies a desired property: *For a cluster C, when all $\boldsymbol{q}^\top \boldsymbol{k}_i$ are close to $\boldsymbol{q}^\top \bar{\boldsymbol{k}}_C$, the cluster is well-approximated and this term gets close to zero. In particular, if $\boldsymbol{q}^\top \boldsymbol{k}_i = \boldsymbol{q}^\top \bar{\boldsymbol{k}}_C, \forall i \in C$, the term becomes exactly zero.*

The first term, $p_C - \widehat{p}_C$, does not satisfy this property. It reflects shift in probability assigned to cluster C, which is coupled with approximation of all clusters due to softmax normalization. Instead, we want to measure how approximating cluster C impacts the overall output. Therefore, we regroup the first term in a source-wise way.

A.3 Regrouping Inter-cluster mass redistribution

Lemma A.1 (Core Lemma: Source-wise regroup of inter-cluster mass redistribution)**.** *Define the auxiliary output*

$$\widetilde{\boldsymbol{o}} = \sum_{C\in\mathcal{C}} \widehat{p}_C \boldsymbol{v}_C, \tag{15}$$

the cluster mass correction factor

$$m_C = \frac{z_C}{\widehat{z}_C}, \tag{16}$$

Then the inter-cluster mass redistribution error admits the exact source-wise decomposition

$$\boxed{\boldsymbol{o} - \widetilde{\boldsymbol{o}} = \sum_{C\in\mathcal{C}} \widehat{p}_C \frac{m_C - 1}{\sum_{C'\in\mathcal{C}} \widehat{p}_{C'} m_{C'}} (\boldsymbol{v}_C - \widetilde{\boldsymbol{o}}).} \tag{17}$$

In particular, if cluster C has no mass approximation error, i.e., $z_C = \widehat{z}_C$ (equivalently, $m_C = 1$), its source-wise contribution to the mass-redistribution error is exactly zero.

Using Lemma A.1, we first rewrite $\boldsymbol{o} - \widetilde{\boldsymbol{o}}$ as:

$$\boldsymbol{o} - \widetilde{\boldsymbol{o}} = \sum_{C \in \mathcal{C}} \widehat{p}_C \frac{m_C - 1}{M} (\boldsymbol{v}_C - \widetilde{\boldsymbol{o}}). \tag{18}$$

With $\|\boldsymbol{v}_C - \widetilde{\boldsymbol{o}}\|_2 \leq 2V_{\max}$:

$$\boxed{\|\boldsymbol{o} - \widetilde{\boldsymbol{o}}\|_2 \leq 2V_{\max} \frac{\sum_{C \in \mathcal{C}} \widehat{p}_C (m_C - 1)}{\sum_{C \in \mathcal{C}} \widehat{p}_C m_C}.} \tag{19}$$

To connect the local mass correction m_C with the approximation fidelity of cluster C, we define the query-dependent logit deviation

$$\delta_C = \max_{i \in C} |s_i - s_C| = \max_{i \in C} \left| \frac{\boldsymbol{q}^\top (\boldsymbol{k}_i - \bar{\boldsymbol{k}}_C)}{\sqrt{d}} \right|. \tag{20}$$

By technical lemma A.2, $m_C \leq \cosh(\delta_C)$. Note that $\sum_{C \in \mathcal{C}} \widehat{p}_C = 1$, which gives

$$\frac{\sum_{C \in \mathcal{C}} \widehat{p}_C (m_C - 1)}{\sum_{C \in \mathcal{C}} \widehat{p}_C m_C} = 1 - \frac{1}{\sum_{C \in \mathcal{C}} \widehat{p}_C m_C}. \tag{21}$$

Since $t \mapsto 1 - 1/t$ is monotonically increasing for $t \geq 1$, we obtain:

$$\boxed{\|\boldsymbol{o} - \widetilde{\boldsymbol{o}}\|_2 \leq 2V_{\max} \frac{\sum_{C \in \mathcal{C}} \widehat{p}_C \left(\cosh(\delta_C) - 1\right)}{\sum_{C \in \mathcal{C}} \widehat{p}_C \cosh(\delta_C)}.} \tag{22}$$

A.4 Intra-cluster value distortion

For the intra-cluster value distortion term, define the exact within-cluster attention weights as

$$w_{i|C} = \frac{e^{s_i}}{z_C}, \qquad \sum_{i \in C} w_{i|C} = 1. \tag{23}$$

Then

$$\boldsymbol{v}_C - \bar{\boldsymbol{v}}_C = \sum_{i \in C} w_{i|C} \boldsymbol{v}_i - \sum_{i \in C} \frac{1}{n_C} \boldsymbol{v}_i \tag{24}$$

$$= \sum_{i \in C} \left(w_{i|C} - \frac{1}{n_C} \right) \boldsymbol{v}_i. \tag{25}$$

Therefore,

$$\|\boldsymbol{v}_C - \bar{\boldsymbol{v}}_C\|_2 \leq \sum_{i \in C} \left| w_{i|C} - \frac{1}{n_C} \right| \|\boldsymbol{v}_i\|_2 \tag{26}$$

$$\leq V_C \sum_{i \in C} \left| w_{i|C} - \frac{1}{n_C} \right|. \tag{27}$$

By Lemma A.3, since $|s_i - s_C| \leq \delta_C$, and

$$w_{i|C} = \frac{e^{s_i - s_C}}{\sum_{j \in C} e^{s_j - s_C}}, \tag{28}$$

we have

$$\sum_{i \in C} \left| w_{i|C} - \frac{1}{n_C} \right| \leq 2 \tanh\left(\frac{\delta_C}{2} \right). \tag{29}$$

Hence,

$$\|\boldsymbol{v}_C - \bar{\boldsymbol{v}}_C\|_2 \leq 2V_C \tanh\left(\frac{\delta_C}{2} \right). \tag{30}$$

Finally,

$$\left\| \sum_{C \in \mathcal{C}} \widehat{p}_C (\boldsymbol{v}_C - \bar{\boldsymbol{v}}_C) \right\|_2 \leq \sum_{C \in \mathcal{C}} \widehat{p}_C \|\boldsymbol{v}_C - \bar{\boldsymbol{v}}_C\|_2 \tag{31}$$

$$\leq \boxed{2 \sum_{C \in \mathcal{C}} \widehat{p}_C V_C \tanh\left(\frac{\delta_C}{2}\right).} \tag{32}$$

Combining Eq. 22 and Eq. 32 via the triangle inequality, we have

$$\|\boldsymbol{o} - \widehat{\boldsymbol{o}}\|_2 \leq \|\boldsymbol{o} - \widetilde{\boldsymbol{o}}\|_2 + \|\widetilde{\boldsymbol{o}} - \widehat{\boldsymbol{o}}\|_2 \tag{33}$$

$$\leq 2V_{\max} \frac{\sum_{C \in \mathcal{C}} \widehat{p}_C \left(\cosh(\delta_C) - 1\right)}{\sum_{C \in \mathcal{C}} \widehat{p}_C \cosh(\delta_C)} + 2 \sum_{C \in \mathcal{C}} \widehat{p}_C V_C \tanh\left(\frac{\delta_C}{2}\right) \tag{34}$$

$$\boxed{= 2 \sum_{C \in \mathcal{C}} \widehat{p}_C \left(V_{\max} \frac{\cosh(\delta_C) - 1}{\sum_{C' \in \mathcal{C}} \widehat{p}_{C'} \cosh(\delta_{C'})} + V_C \tanh\left(\frac{\delta_C}{2}\right) \right).} \tag{35}$$

The maximum logit deviation δ_C, however, cannot be used directly for access decisions, since evaluating it requires inspecting all keys within cluster C, defeating the purpose of cluster-level selection. We therefore construct an efficiently computable query-dependent upper bound δ_C^{bound} from compact cluster statistics, such that

$$\delta_C \leq \delta_C^{\text{bound}}. \tag{36}$$

The specific construction of δ_C^{bound} is deferred to Appendix A.7. Since the above error bound is monotonically non-decreasing with respect to each δ_C, replacing δ_C with δ_C^{bound} yields a computable upper bound:

$$\|\boldsymbol{o} - \widehat{\boldsymbol{o}}\|_2 \leq 2 \sum_{C \in \mathcal{C}} \underbrace{\widehat{p}_C \left(V_{\max} \frac{\cosh(\delta_C^{\text{bound}}) - 1}{\sum_{C' \in \mathcal{C}} \widehat{p}_{C'} \cosh(\delta_{C'}^{\text{bound}})} + V_C \tanh\left(\frac{\delta_C^{\text{bound}}}{2}\right) \right)}_{\text{Risk Score } R_C \text{ for cluster } C}. \tag{37}$$

We write the resulting upper bound in a clusterwise form and use each clusterwise term as its risk score R_C.

A.5 Proof of core lemma

Proof. We define absolute mass correction of cluster C:

$$\Delta z_C = z_C - \widehat{z}_C \geq 0 \tag{38}$$

Proof of inequality can be done using Jensen's inequality with $1/n_C \sum s_i = s_C$. This revels that approximation underestimate the mass of cluster.

We have:

$$p_C - \widehat{p}_C = \frac{z_C}{Z} - \widehat{p}_C \tag{39}$$

$$= \frac{\Delta z_C + \widehat{z}_C - \widehat{p}_C Z}{Z} \tag{40}$$

$$= \frac{\Delta z_C + \widehat{p}_C \widehat{Z} - \widehat{p}_C Z}{Z} \tag{41}$$

$$= \underbrace{\frac{\Delta z_C}{Z}}_{\text{effect of cluster } C} - \underbrace{\frac{\widehat{p}_C \sum_{C'} \Delta z_{C'}}{Z}}_{\text{effect of changed gobal normlization}}. \tag{42}$$

If we define $T_{C,C'}$ as the effect of approximating C' on cluster C:

$$T_{C,C'} = \frac{\Delta z_{C'}}{Z} (\mathbf{1}_{[C=C']} - \widehat{p}_C). \tag{43}$$

Naturally we have:

$$p_C - \widehat{p}_C = \sum_{C'} T_{C,C'} \tag{44}$$

And:

$$\boldsymbol{o} - \widetilde{\boldsymbol{o}} = \sum_C (p_C - \widehat{p}_C)\boldsymbol{v}_C = \sum_C \sum_{C'} T_{C,C'}\boldsymbol{v}_C \tag{45}$$

As analysed before, we want to know how much output distortion originates from approximating a particular cluster:

$$\begin{aligned}
\boldsymbol{o} - \widetilde{\boldsymbol{o}} &= \sum_{C'} \sum_C T_{C,C'}\boldsymbol{v}_C && (46)\\
&= \sum_{C'} \sum_C \frac{\Delta z_{C'}}{Z}(\mathbf{1}_{[C=C']} - \widehat{p}_C)\boldsymbol{v}_C && (47)\\
&= \sum_{C'} \frac{\Delta z_{C'}}{Z}\left(\boldsymbol{v}_{C'} - \sum_C \widehat{p}_C \boldsymbol{v}_C\right) && (48)\\
&= \sum_{C'} \frac{\Delta z_{C'}}{Z}(\boldsymbol{v}_{C'} - \widetilde{\boldsymbol{o}}) && (49)
\end{aligned}$$

We define mass correction factor m_C as

$$m_C = \frac{z_C}{\widehat{z}_C} = \frac{1}{n_C}\sum_{i \in C} e^{s_i - s_C} \geq 1 \tag{50}$$

The total mass correction factor M can be defined as:

$$M = \frac{Z}{\widehat{Z}} = \frac{\sum_{C \in \mathcal{C}} z_C}{\widehat{Z}} = \frac{\sum_{C \in \mathcal{C}} m_C \widehat{z}_C}{\widehat{Z}} = \sum_{C \in \mathcal{C}} \widehat{p}_C m_C \tag{51}$$

We have:

$$\begin{aligned}
\boldsymbol{o} - \widetilde{\boldsymbol{o}} &= \sum_C \frac{\Delta z_C}{Z}(\boldsymbol{v}_C - \widetilde{\boldsymbol{o}}) && (52)\\
&= \sum_C \frac{\widehat{z}_C(m_C - 1)}{\widehat{Z}M}(\boldsymbol{v}_C - \widetilde{\boldsymbol{o}}) && (53)\\
&= \sum_C \frac{\widehat{p}_C(m_C - 1)}{M}(\boldsymbol{v}_C - \widetilde{\boldsymbol{o}}) && (54)
\end{aligned}$$

□

A.6 Proof of technical lemmas

Lemma A.2 (Cluster Mass Correction Bound)**.** *For any cluster $C \in \mathcal{C}$, let the query-dependent logit deviation be defined as*

$$\delta_C \coloneqq \max_{i \in C} |s_i - s_C| = \max_{i \in C} \left| \frac{\boldsymbol{q}^\top (\boldsymbol{k}_i - \bar{\boldsymbol{k}}_C)}{\sqrt{d}} \right|. \tag{55}$$

Then, the local mass correction factor $m_C \coloneqq \frac{1}{n_C}\sum_{i \in C} e^{s_i - s_C}$ satisfies

$$m_C \leq \cosh(\delta_C). \tag{56}$$

Proof. Let $x_i = s_i - s_C$. By definition of s_C, the mean of $\{x_i\}_{i \in C}$ vanishes:

$$\frac{1}{n_C}\sum_{i \in C} x_i = \frac{1}{n_C}\sum_{i \in C} s_i - s_C = 0. \tag{57}$$

Moreover, by definition of δ_C, we have $x_i \in [-\delta_C, \delta_C]$ for all $i \in C$. By the convexity of the exponential function $x \mapsto e^x$, any $x \in [-\delta_C, \delta_C]$ can be upper-bounded via the secant line between the endpoints:

$$e^x \leq \frac{\delta_C + x}{2\delta_C} e^{\delta_C} + \frac{\delta_C - x}{2\delta_C} e^{-\delta_C}. \tag{58}$$

Averaging this inequality over all tokens in cluster C, we obtain

$$\begin{aligned} m_C &= \frac{1}{n_C} \sum_{i \in C} e^{x_i} && (59)\\ &\leq \frac{1}{n_C} \sum_{i \in C} \left[\frac{\delta_C + x_i}{2\delta_C} e^{\delta_C} + \frac{\delta_C - x_i}{2\delta_C} e^{-\delta_C} \right] && (60)\\ &= \frac{\delta_C + \frac{1}{n_C}\sum_{i \in C} x_i}{2\delta_C} e^{\delta_C} + \frac{\delta_C - \frac{1}{n_C}\sum_{i \in C} x_i}{2\delta_C} e^{-\delta_C} && (61)\\ &= \frac{e^{\delta_C} + e^{-\delta_C}}{2} && (62)\\ &= \cosh(\delta_C), && (63) \end{aligned}$$

which completes the proof. □

Lemma A.3 (Softmax deviation from uniform)**.** *Let $x_1, \ldots, x_n \in \mathbb{R}$ satisfy*

$$|x_i| \leq \delta, \qquad i = 1, \ldots, n, \tag{64}$$

and define

$$w_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}. \tag{65}$$

Then

$$\boxed{\sum_{i=1}^n \left| w_i - \frac{1}{n} \right| \leq 2 \tanh\left(\frac{\delta}{2}\right).} \tag{66}$$

Proof. Since $x_i \in [-\delta, \delta]$, for any i, j,

$$\frac{w_i}{w_j} = e^{x_i - x_j} \leq e^{2\delta}. \tag{67}$$

Let

$$r_i = n w_i, \tag{68}$$

so that

$$\frac{1}{n} \sum_{i=1}^n r_i = 1. \tag{69}$$

Denote

$$a = \min_i r_i, \qquad b = \max_i r_i. \tag{70}$$

Then

$$0 < a \leq 1 \leq b, \qquad \frac{b}{a} \leq e^{2\delta}. \tag{71}$$

For any random variable $R \in [a, b]$ with $\mathbb{E}[R] = 1$, the convexity of $|R - 1|$ implies

$$\frac{1}{2}\mathbb{E}|R - 1| \leq \frac{(1-a)(b-1)}{b-a}. \tag{72}$$

Applying this to the empirical distribution over $\{r_i\}_{i=1}^n$ gives

$$\frac{1}{2} \sum_{i=1}^n \left| w_i - \frac{1}{n} \right| \leq \frac{(1-a)(b-1)}{b-a}. \tag{73}$$

For a fixed ratio $R = b/a$, the right-hand side is maximized at

$$a = \frac{1}{\sqrt{R}}, \qquad b = \sqrt{R}, \tag{74}$$

which yields

$$\frac{(1-a)(b-1)}{b-a} \le \frac{\sqrt{R}-1}{\sqrt{R}+1}. \tag{75}$$

Since $R \le e^{2\delta}$,

$$\frac{\sqrt{R}-1}{\sqrt{R}+1} \le \frac{e^{\delta}-1}{e^{\delta}+1} = \tanh\left(\frac{\delta}{2}\right). \tag{76}$$

Therefore,

$$\sum_{i=1}^{n} \left| w_i - \frac{1}{n} \right| \le 2\tanh\left(\frac{\delta}{2}\right). \tag{77}$$

□

A.7 Computable upper bound for δ_C.

The quantity δ_C directly measures the worst-case logit deviation within cluster C under the current query:

$$\delta_C = \max_{i \in C} \left| \frac{\boldsymbol{q}^\top (\boldsymbol{k}_i - \bar{\boldsymbol{k}}_C)}{\sqrt{d}} \right|. \tag{78}$$

While δ_C provides a natural measure of local fidelity, computing it exactly requires accessing all keys in the cluster, which defeats the purpose of efficient selection. We therefore seek a query-dependent upper bound $D_C(\boldsymbol{q})$ such that

$$\boxed{\delta_C^{\text{bound}}(\boldsymbol{q}) \ge \delta_C.} \tag{79}$$

A.7.1 Axis-aligned box bound.

Let $\boldsymbol{k}_C^{\min}$ and $\boldsymbol{k}_C^{\max}$ be the coordinate-wise minimum and maximum keys in cluster C, and define

$$s_C = \frac{\boldsymbol{q}^\top \bar{\boldsymbol{k}}_C}{\sqrt{d}}. \tag{80}$$

Since every $\boldsymbol{k}_i$, $i \in C$, lies inside the box $[\boldsymbol{k}_C^{\min}, \boldsymbol{k}_C^{\max}]$, its query-key score is bounded by

$$s_C^{\text{upper}} = \frac{1}{\sqrt{d}} \sum_{j=1}^{d} \max\left\{ q_j k_{C,j}^{\min}, q_j k_{C,j}^{\max} \right\}, \tag{81}$$

$$s_C^{\text{lower}} = \frac{1}{\sqrt{d}} \sum_{j=1}^{d} \min\left\{ q_j k_{C,j}^{\min}, q_j k_{C,j}^{\max} \right\}. \tag{82}$$

Therefore,

$$\boxed{\delta_C^{\text{box}}(\boldsymbol{q}) = \max\left\{ |s_C^{\text{upper}} - s_C|, |s_C^{\text{lower}} - s_C| \right\} \ge \delta_C.} \tag{83}$$

This bound requires two d-dimensional vectors per cluster and $O(d)$ query-time computation. It can be loose because its coordinate-wise extrema may come from different keys.

A.7.2 Diagonal-Ellipsoid Bound

For a centered key $\boldsymbol{r}_i = \boldsymbol{k}_i - \bar{\boldsymbol{k}}_C$, define the coordinate-wise radii

$$w_{C,j} = \max_{i \in C} |r_{i,j}|. \tag{84}$$

For clarity, we assume $w_{C,j} > 0$; a zero-width coordinate has zero residual for every key and can be omitted without affecting the bound. We further define

$$\rho_C = \max_{i \in C} \left(\sum_{j=1}^{d_k} \frac{r_{i,j}^2}{w_{C,j}^2} \right)^{1/2}. \tag{85}$$

The vector $\boldsymbol{w}_C$ normalizes the scale of each coordinate, while ρ_C measures the largest residual after this normalization. Equivalently, all centered keys lie inside the diagonal ellipsoid

$$\sum_{j=1}^{d_k} \frac{r_j^2}{w_{C,j}^2} \le \rho_C^2. \tag{86}$$

The resulting query-dependent bound is

$$\boxed{\delta_C^{\mathrm{ell}}(\boldsymbol{q}) = \frac{\rho_C}{\sqrt{d_k}} \left(\sum_{j=1}^{d_k} q_j^2 w_{C,j}^2 \right)^{1/2}}. \tag{87}$$

Indeed, for every $i \in C$, weighted Cauchy–Schwarz gives

$$|\boldsymbol{q}^\top \boldsymbol{r}_i| = \left| \sum_{j=1}^{d_k} (q_j w_{C,j}) \frac{r_{i,j}}{w_{C,j}} \right| \tag{88}$$

$$\le \left(\sum_{j=1}^{d_k} q_j^2 w_{C,j}^2 \right)^{1/2} \left(\sum_{j=1}^{d_k} \frac{r_{i,j}^2}{w_{C,j}^2} \right)^{1/2} \tag{89}$$

$$\le \rho_C \left(\sum_{j=1}^{d_k} q_j^2 w_{C,j}^2 \right)^{1/2}. \tag{90}$$

Therefore,

$$\delta_C^{\mathrm{ell}}(\boldsymbol{q}) \ge \max_{i \in C} \frac{|\boldsymbol{q}^\top (\boldsymbol{k}_i - \bar{\boldsymbol{k}}_C)|}{\sqrt{d_k}}. \tag{91}$$

Both $\boldsymbol{w}_C$ and ρ_C are query-independent and can be precomputed once per block. The bound stores one vector and one scalar per block and takes $O(d_k)$ operations to evaluate for each query–block pair.

B REFINEKV DETAILS

B.1 Q-ANCHOR

Prompt-Query Anchors. During prefill, we use the query vectors from the last Q prompt positions as anchors and cache the clusters they select. These anchor-selected clusters remain expanded throughout decoding, in addition to those selected by the current query. We use $Q = 32$ for RULER and HELMET and disable query anchors for LongBench. The anchor mask is computed once during prefill and reused throughout decoding. Table 3 quantifies the resulting quality–latency trade-off.

Ablation for Q-anchor. We additionally perform abletion on the number of query anchors. The results are shown in Table 3.

B.2 KERNEL IMPLEMENTATION

We implement REFINEKV using custom Triton kernels. After prefill, the prompt K/V cache is partitioned into blocks of 32 tokens. For each block, we compute and cache its key and value representatives, diagonal-ellipsoid bound statistics, maximum value norm, and valid-token count.

Table 3: Query-anchor ablation of Llama-3.1-8B-Instruct on the fixed RULER-128K evaluation subset. All sparse variants use the same default configuration and differ only in the number of query anchors Q; $Q = 32$ is our default setting. Latency is steady-state TPOT measured under the aligned wall-clock protocol. Bold identify the default setting.

Metric	Ref.	Number of query anchors Q			
	Full	0	16	**32** (default)	64
Acc.↑	73.35	67.01	70.03	**70.94**	72.43
Decode budget (%)↓	100.00	9.03	15.68	**19.41**	25.11
Steady TPOT (ms)↓	26.52	15.19	15.92	**16.35**	17.02
Speedup↑	1.00×	1.75×	1.67×	**1.62**×	1.56×

Prompt-anchor queries are captured during prefill, and their selected blocks are computed once and stored as a compact per-KV-head mask. These summaries and masks are reused throughout generation.

At each decoding step, the block-selection kernel jointly processes the query heads associated with each KV head. It computes the representative attention score and deviation bound for every block, applies the expansion criterion of REFINEKV, and combines the resulting decisions with the cached prompt-anchor mask. The output is a compact selection mask that remains on the GPU. The sparse-attention kernel then reads token-level K/V directly from the original cache only for selected blocks; unselected blocks contribute through their cached representatives, while generated suffix tokens are always processed exactly. The three types of contributions are merged using online softmax, with BF16 Tensor-Core operations for matrix products and FP32 accumulation for numerical stability. We use a preallocated static KV cache and reuse all intermediate workspaces across tokens. For fixed-length latency evaluation, the complete decoding step is captured in a CUDA Graph, reducing Python and kernel-launch overhead to one graph replay per token.

C EXPERIMENT DETAILS

C.1 CRITICAL KV ANALYSIS

Top-p KV analysis. For the top-p analysis, we evaluate all 32 layers of Llama-3.1-8B on 20 non-overlapping 1,024-token segments from the WikiText-2 test split. For each query head, we select the smallest KV subset that accumulates attention mass $p \in \{0, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99, 1\}$. Because the model uses GQA, the four query heads sharing a KV head access the union of their selected subsets. We report the average per-query KV percentage, the union across all queries, and the union across distant queries satisfying $q - k \geq 64$. Output error is measured after renormalizing attention over the selected KVs as $\|O_p - O_{\text{full}}\|_2 / \|O_{\text{full}}\|_2$.

Perplexity evaluation. For the perplexity experiment, we use 20 non-overlapping 2,048-token WikiText-2 segments, totaling 40,940 next-token predictions. We compare our query-adaptive method against oracle top-k and the query-independent H2O, RoCo, and TOVA eviction methods. The horizontal axis reports the measured fraction of KV entries accessed relative to full causal attention, rather than the nominal cache budget. All results are averaged across samples, and error bars show 95% confidence intervals.

C.2 MAIN EXPERIMENT SETUP

C.2.1 MODELS

We evaluate `Llama-3.1-8B-Instruct` (Grattafiori et al., 2024) and `Qwen2.5-7B-Instruct` (Qwen et al., 2025) without additional training or fine-tuning. Llama-3.1-8B contains 32 transformer layers with 32 query heads and 8 KV heads, corresponding to four query heads per KV head. It uses the rotary position encoding (Su et al., 2023) with a native context length of 128K.

Qwen2.5-7B contains 28 layers with 28 query heads and 4 KV heads, resulting in a larger GQA group of seven query heads per KV head. Qwen2.5 natively supports a 32K context window with its default RoPE configuration. For the 64K and 128K evaluations, we follow the officially recommended long-context setting by enabling static YaRN(Peng et al., 2026) with a scaling factor of four and an original context length of 32K. This positional-encoding extension is applied identically to all methods. Both models have a head dimension of 128 and are evaluated in BF16 with batch size one and greedy decoding.

C.2.2 BENCHMARKS

For LongBench (Bai et al., 2024), we use the 16-task English/code subset consisting of 14 English tasks, `lcc`, and `repobench-p`, with 3,750 test examples per configuration. Quality is evaluated on LongBench using the official prompts, generation limits, and task-specific metrics, while latency is measured separately using standardized 128K prompts.

For RULER (Hsieh et al., 2024), we evaluate all 13 official tasks at context length of 128K, using the same 20 source examples per task and length, for a total of 260 examples per configuration. RULER data are generated separately for each model and tokenizer.

For HELMET (Yen et al., 2025), we include all tasks with objective automatic evaluation and exclude only subjective tasks that require an LLM-as-a-judge. We follow the official task-specific formatting, generation limits, stopping rules, and evaluation metrics. For all benchmarks, we compute each task score independently and report the unweighted macro-average across tasks.

C.2.3 BASELINES

Full Attention. Full Attention retains the complete visible KV cache and serves as the reference for both quality and latency. We use the model's native GQA layout and optimized scaled dot-product attention without replicating KV heads. The latency implementation uses a preallocated static KV cache and eager decoding under the same BF16 and batch-size-one protocol as the efficient baselines.

SnapKV. We use the KVPress (Devoto et al., 2025) implementation of SnapKV with an observation window of 64 tokens and a pooling kernel size of 5. We evaluate target KV-retention ratios of 10%, 25%, and 50%, where the percentage denotes the fraction of prompt KV states retained after prefill compression. Decode attention operates directly on the compressed cache, while newly generated tokens are retained.

StreamLLM. We use the KVPress (Devoto et al., 2025) implementation of StreamLLM with four attention-sink tokens. After preserving these sink tokens, the remaining budget is assigned to the most recent prompt tokens. We evaluate prompt-KV retention ratios of 10%, 25%, and 50%. The prompt cache is compressed after prefill, and all newly generated tokens are retained during decoding.

PyramidKV. We configure PyramidKV with an observation window of 64 tokens, a pooling kernel size of 5, and $\beta = 20$. We evaluate mean retention targets of 10%, 25%, and 50%; because PyramidKV distributes its budget non-uniformly across layers, these percentages denote the configured average retention rather than an identical per-layer budget. The selected prompt KVs are stored in a visible-prefix static cache and processed using eager native-GQA attention.

Quest. We use a page size of 16 tokens and evaluate absolute token budgets of 512, 1,024, 2,048, and 4,096 tokens. Following the official setting, the first two transformer layers use Full Attention, while the remaining layers apply query-dependent page selection. The mandatory tail page is included in the reported token budget. We use paged CUDA decoding for Llama and a native-head parallel page-scoring implementation for Qwen, without physically replicating GQA KV heads.

Double-P. We use clusters of 32 prompt tokens, four iterations of per-KV-head k-means, four exact sink tokens, and a 64-token exact local window. We evaluate the threshold pairs $(p_1, p_2) \in \{(0.85, 0.50), (0.85, 0.70), (0.90, 0.60), (0.95, 0.70)\}$ when complete measurements are available.

Clusters outside the p_1 probability prefix are pruned, clusters between the p_1 and p_2 prefixes are represented by their centroids, and clusters inside the p_2 prefix are expanded to exact tokens. Prompt clustering is performed once after prefill, and generated tokens remain exact. Since the authors' production kernels are *not publicly available*, we use our optimized Triton reimplementation and do not describe it as the official Double-P implementation.

C.2.4 METRICS.

We focus on steady-state decoding because REFINEKV primarily targets the repeated KV-cache access during autoregressive generation. Prefill processes all prompt tokens in parallel through regular, highly optimized matrix–matrix multiplications, whereas batch-one decoding executes sequentially for each generated token and is often dominated by memory-intensive KV-cache access. Prefill is therefore a one-time, highly parallelized cost, while decode latency accumulates with the output length. As shown in Table 4, our auxiliary experiment on Llama-3.1-8B shows that decoding accounts for 82.2% with 32K prefill/512 decode setting, and 90.3% with 32K/1024 setting. We thus evaluate serving efficiency using decode TPOT and treat prefill latency and time to first token as separate costs.

Table 4: Latency breakdown of prefill and decoding on Llama-3.1-8B in BF16. Each result is averaged over three runs.

Sequence length		Latency (s)		Fraction (%)	
Prefill	Decode	Prefill	Decode	Prefill	Decode
16K	512	1.01	9.20	9.9	90.1
32K	512	2.42	11.15	17.8	82.2
32K	1024	2.41	22.37	9.7	90.3

For every method, we disable early stopping and generate exactly 128 output tokens. We exclude the output token produced during prefill, as it does not require an autoregressive decoding step, and discard the first post-prefill iteration to remove one-time initialization overhead. Let T_{steady} denote the elapsed time of the remaining 127 decoding iterations. We compute

$$\text{TPOT} = \frac{T_{\text{steady}}}{127} \text{ ms/token}.$$

The timed region includes all method-specific operations, including token selection or routing, attention computation, Transformer execution, KV-cache updates, and generation-loop overhead, while excluding model loading, compilation, warm-up, and prefill.

Table 5: Sensitivity of steady-state TPOT to the generation length. Values in parentheses denote the relative change from 128 tokens. All lengths use identical prompts, cache capacities, and method configurations.

	TPOT (ms/token)		
Benchmark	128 tokens	256 tokens	512 tokens
LongBench	14.76	14.61 (−1.03%)	14.55 (−1.43%)
RULER	16.35	16.20 (−0.92%)	16.14 (−1.27%)
HELMET	17.20	17.06 (−0.80%)	17.01 (−1.07%)

Choice of generation length. We report latency using 128 generated tokens. As shown in Table 5, increasing the generation length to 256 or 512 tokens changes steady-state TPOT by at most 1.43% on average across all three benchmarks; the largest per-prompt change is 1.69%, within our predefined 3% practical threshold. Thus, 128 tokens are sufficient to obtain a stable estimate of steady-state decoding efficiency, while longer generations substantially increase evaluation cost without materially changing the conclusion. We therefore use 128 tokens in the main experiments.

C.2.5 HARDWARE SETTINGS.

All experiments are conducted on NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs. Each latency measurement uses a single exclusive GPU. Unless noted otherwise, all methods use bfloat16 and batch size one.

Table 6: LongBench per-task quality–latency comparison. Each cell reports ACC/AccL for one original benchmark task, where ACC is that task's benchmark score and AccL is latency acceleration relative to Full, defined as Full steady TPOT divided by method steady TPOT for the same model and task; higher is better for both. Shaded cells are Pareto-optimal among the 11 displayed fixed operating points for that task after requiring AccL ≥ 1 and an absolute ACC drop versus task-matched Full of at most 5 points. Underlined non-Full cells have an absolute ACC drop versus task-matched Full of less than 0.3 point.

	Method	S-Doc QA			M-Doc QA			Summ.			Few-shot			Synth.		Code	
		NarrativeQA	Qasper	MultiFieldQA	HotpotQA	2WikiMQA	MuSiQue	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	Count	Retrieval	LCC	RepoBench-P
Llama	Full	29.6/1.0	45.5/1.0	55.1/1.0	56.5/1.0	45.1/1.0	31.2/1.0	35.0/1.0	25.2/1.0	27.2/1.0	72.5/1.0	91.5/1.0	43.6/1.0	7.9/1.0	99.5/1.0	62.9/1.0	56.7/1.0
	Snap 10%	30.3/1.7	36.5/1.7	52.4/1.7	54.9/1.7	42.7/1.7	30.8/1.7	27.0/1.7	23.4/1.7	19.8/1.7	63.0/1.7	91.9/1.7	42.8/1.7	7.2/1.7	99.5/1.7	60.8/1.7	56.3/1.7
	Snap 25%	30.0/1.5	42.2/1.5	54.2/1.5	55.1/1.5	44.4/1.5	31.1/1.5	30.4/1.5	25.0/1.5	23.4/1.5	68.5/1.5	91.6/1.5	42.4/1.5	6.9/1.5	99.5/1.5	62.8/1.5	56.8/1.5
	Str. 10%	26.3/1.7	26.2/1.7	33.7/1.7	47.6/1.7	37.0/1.7	25.2/1.7	25.4/1.7	21.4/1.7	19.7/1.7	58.5/1.7	88.8/1.7	42.5/1.7	6.4/1.7	88.5/1.7	59.4/1.7	52.9/1.7
	Str. 25%	28.1/1.5	31.3/1.5	35.6/1.5	50.4/1.5	40.9/1.5	25.3/1.5	29.1/1.5	22.0/1.5	23.0/1.5	66.0/1.5	90.0/1.5	42.4/1.5	7.9/1.5	84.5/1.5	60.9/1.5	55.3/1.5
	Pyr. 10%	30.3/1.7	36.0/1.7	52.3/1.7	54.8/1.7	42.7/1.7	31.6/1.7	26.7/1.7	24.2/1.7	19.9/1.7	63.0/1.7	91.9/1.7	43.0/1.7	7.2/1.7	99.5/1.7	60.6/1.7	56.1/1.7
	Pyr. 25%	30.5/1.5	42.5/1.5	53.8/1.5	55.4/1.5	44.4/1.5	31.6/1.5	30.1/1.5	24.9/1.5	23.6/1.5	67.5/1.5	91.5/1.5	43.6/1.5	6.4/1.5	99.5/1.5	62.8/1.5	56.9/1.5
	Quest	30.5/0.9	45.1/0.9	55.5/0.9	55.3/0.9	45.9/0.9	32.8/0.9	35.8/0.9	25.8/0.9	27.2/0.9	72.0/0.9	91.5/0.9	43.1/0.9	6.9/0.9	99.5/0.9	62.9/0.9	56.4/0.9
	Double	30.6/1.2	43.8/1.2	55.9/1.2	54.7/1.2	43.6/1.2	31.4/1.2	34.6/1.2	24.5/1.2	26.7/1.2	71.5/1.2	91.6/1.2	42.9/1.2	7.0/1.2	99.0/1.2	63.5/1.2	55.6/1.2
	Our τ.01	**30.3/1.8**	**45.0/1.8**	**55.2/1.8**	**55.1/1.8**	**44.6/1.8**	**31.3/1.8**	**35.1/1.8**	**25.1/1.8**	**27.2/1.8**	**72.5/1.7**	**91.6/1.7**	**44.4/1.7**	**6.9/1.8**	**100.0/1.8**	**63.0/1.8**	**56.8/1.8**
	Our τ.1	**29.5/1.8**	**43.9/1.8**	**55.5/1.8**	**54.8/1.8**	**44.3/1.8**	**32.0/1.8**	**33.5/1.8**	**24.7/1.8**	**27.0/1.8**	**68.0/1.8**	**91.5/1.8**	**43.4/1.8**	**6.3/1.8**	**98.5/1.8**	**62.7/1.8**	**56.1/1.8**
Qwen	Full	29.3/1.0	43.5/1.0	52.1/1.0	58.7/1.0	47.6/1.0	30.0/1.0	31.8/1.0	23.6/1.0	24.0/1.0	72.5/1.0	89.4/1.0	45.7/1.0	9.0/1.0	100.0/1.0	60.1/1.0	67.1/1.0
	Snap 10%	29.2/1.3	36.1/1.3	48.8/1.3	57.4/1.3	46.4/1.3	29.8/1.3	24.8/1.3	22.7/1.3	16.4/1.3	54.5/1.3	88.4/1.3	43.8/1.3	9.0/1.3	100.0/1.3	53.4/1.3	65.5/1.3
	Snap 25%	29.9/1.2	41.1/1.2	50.7/1.2	58.3/1.2	46.1/1.3	30.6/1.3	28.4/1.3	23.3/1.2	20.1/1.3	65.0/1.2	88.8/1.3	44.9/1.2	9.0/1.2	100.0/1.3	57.2/1.3	66.7/1.3
	Str. 10%	23.4/1.3	27.1/1.3	28.2/1.3	43.9/1.3	38.0/1.3	19.7/1.3	23.8/1.3	19.3/1.3	16.0/1.3	58.5/1.3	86.6/1.3	43.3/1.3	9.5/1.3	37.0/1.3	49.9/1.3	57.7/1.3
	Str. 25%	25.9/1.2	29.9/1.2	31.3/1.3	47.5/1.3	40.7/1.2	22.8/1.2	27.7/1.2	20.1/1.3	19.5/1.2	64.5/1.3	89.1/1.2	44.0/1.2	9.0/1.3	57.2/1.2	53.4/1.2	62.3/1.2
	Pyr. 10%	29.0/1.3	37.7/1.3	48.6/1.3	55.5/1.3	44.7/1.3	27.4/1.3	25.7/1.3	22.8/1.3	18.0/1.3	55.0/1.3	85.4/1.3	45.2/1.3	9.0/1.3	100.0/1.3	56.5/1.3	63.5/1.3
	Pyr. 25%	29.9/1.3	41.5/1.3	49.3/1.3	56.9/1.3	45.6/1.3	29.5/1.3	28.1/1.3	23.2/1.3	21.5/1.3	67.0/1.3	87.0/1.3	46.6/1.3	9.0/1.3	100.0/1.3	60.3/1.3	63.9/1.3
	Quest	29.0/1.0	43.7/1.0	52.3/1.0	57.4/1.0	46.3/1.0	28.9/1.0	33.1/1.0	23.4/1.0	24.3/1.0	72.0/1.0	89.6/1.0	45.9/1.0	7.5/1.0	99.5/1.0	59.5/1.0	65.5/1.0
	Double	28.5/0.9	43.8/0.9	50.7/0.9	58.3/0.9	44.7/0.9	28.2/0.9	33.1/0.9	23.3/0.9	25.5/0.9	73.5/0.9	88.8/0.9	46.8/0.9	9.0/0.9	98.0/0.9	60.6/0.9	64.2/0.9
	Our τ.01	**28.8/1.3**	**44.7/1.3**	**51.3/1.3**	**58.1/1.3**	**45.5/1.3**	**29.7/1.3**	**33.5/1.3**	**24.0/1.3**	**25.6/1.3**	**73.5/1.2**	**88.9/1.2**	**46.8/1.3**	**9.0/1.3**	**100.0/1.3**	**61.5/1.3**	**66.9/1.3**
	Our τ.1	**29.7/1.4**	**45.2/1.4**	**50.4/1.4**	**57.9/1.4**	**45.4/1.4**	**29.7/1.4**	**33.9/1.4**	**24.6/1.4**	**25.6/1.4**	**73.5/1.4**	**88.9/1.4**	**47.2/1.4**	**9.0/1.4**	**99.0/1.4**	**61.1/1.4**	**66.0/1.4**

C.3 Extra Experiment Results

C.3.1 Main table tasks

For compact presentation, we aggregate semantically related tasks into task groups; no tasks are cherry-picked or omitted, and every task is included exactly once. For LongBench, Doc QA combines single-document and multi-document question answering, Summ. denotes long-document summarization, Few-shot denotes few-shot learning, Synth. covers synthetic counting and retrieval, and Code evaluates long-context code understanding and completion. For RULER, NIAH combines single-key, multi-key, multi-value, and multi-query needle-in-a-haystack retrieval, A&T denotes aggregation and tracking, and QA denotes question answering. For HELMET, RAG denotes retrieval-augmented generation, LC-QA denotes long-context question answering, ICL denotes in-context learning, and Rec./Rank. combines recall and reranking tasks. We put the detail for subtask result in: Table 6, Table 7 and Table 8.

Table 7: RULER per-task quality–latency comparison.

	Method	S-NIAH			MK-NIAH			MV/MQ-NIAH		A&T			QA	
		NIAH-S1	NIAH-S2	NIAH-S3	NIAH-MK1	NIAH-MK2	NIAH-MK3	NIAH-MV	NIAH-MQ	CWE	FWE	VT	QA-1	QA-2
Llama	Full	100.0/1.0	100.0/1.0	100.0/1.0	90.0/1.0	70.0/1.0	55.0/1.0	92.5/1.0	100.0/1.0	0.0/1.0	70.0/1.0	66.0/1.0	65.0/1.0	45.0/1.0
	Snap 10%	100.0/1.7	100.0/1.7	70.0/1.7	90.0/1.7	60.0/1.7	30.0/1.7	88.8/1.7	97.5/1.7	0.5/1.7	61.7/1.7	60.0/1.7	65.0/1.7	45.0/1.7
	Snap 25%	100.0/1.5	100.0/1.5	85.0/1.5	90.0/1.5	65.0/1.5	40.0/1.5	93.8/1.5	100.0/1.5	0.0/1.5	65.0/1.5	59.0/1.5	65.0/1.5	45.0/1.5
	Str. 10%	5.0/1.7	20.0/1.7	5.0/1.7	10.0/1.7	10.0/1.7	20.0/1.7	10.0/1.7	10.0/1.7	0.5/1.7	80.0/1.7	9.0/1.7	50.0/1.7	25.0/1.7
	Str. 25%	35.0/1.5	45.0/1.5	20.0/1.5	40.0/1.5	30.0/1.5	25.0/1.5	25.0/1.5	25.0/1.5	1.0/1.5	76.7/1.5	25.0/1.5	65.0/1.5	35.0/1.5
	Pyr. 10%	100.0/1.7	100.0/1.7	70.0/1.7	90.0/1.7	60.0/1.7	35.0/1.7	83.8/1.7	98.8/1.7	0.5/1.7	55.0/1.7	62.0/1.7	65.0/1.7	45.0/1.7
	Pyr. 25%	100.0/1.5	100.0/1.5	85.0/1.5	90.0/1.5	60.0/1.5	50.0/1.5	93.8/1.5	100.0/1.5	0.0/1.5	63.3/1.5	62.0/1.5	65.0/1.5	45.0/1.5
	Quest	100.0/0.9	100.0/0.9	75.0/0.9	90.0/0.9	45.0/0.9	0.0/0.9	77.5/0.9	87.5/0.9	0.0/0.9	66.7/0.9	64.0/0.9	65.0/0.9	40.0/0.9
	Double	95.0/1.2	80.0/1.2	95.0/1.2	85.0/1.2	20.0/1.2	5.0/1.2	42.5/1.2	53.8/1.2	0.5/1.2	68.3/1.2	34.0/1.2	65.0/1.2	45.0/1.2
	Our τ.01	**100.0/1.7**	**100.0/1.5**	**100.0/1.5**	**90.0/1.5**	**65.0/1.6**	**40.0/1.6**	**87.5/1.6**	**93.8/1.6**	**0.0/1.6**	**71.7/1.6**	**68.0/1.7**	**65.0/1.5**	**45.0/1.5**
	Our τ.1	**100.0/1.8**	**95.0/1.8**	**95.0/1.8**	**95.0/1.8**	**40.0/1.8**	**0.0/1.8**	**51.2/1.8**	**87.5/1.8**	**1.0/1.8**	**60.0/1.8**	**59.0/1.8**	**65.0/1.8**	**45.0/1.8**
Qwen	Full	100.0/1.0	100.0/1.0	95.0/1.0	85.0/1.0	45.0/1.0	10.0/1.0	70.0/1.0	85.0/1.0	40.5/1.0	48.3/1.0	82.0/1.0	60.0/1.0	25.0/1.0
	Snap 10%	100.0/1.3	95.0/1.3	0.0/1.3	75.0/1.3	0.0/1.3	0.0/1.3	20.0/1.3	72.5/1.3	35.5/1.3	43.3/1.3	82.0/1.3	55.0/1.3	20.0/1.3
	Snap 25%	100.0/1.2	100.0/1.3	5.0/1.3	70.0/1.3	5.0/1.2	0.0/1.2	43.8/1.2	86.2/1.3	44.0/1.2	46.7/1.2	83.0/1.2	55.0/1.2	25.0/1.3
	Str. 10%	5.0/1.3	20.0/1.3	5.0/1.3	5.0/1.3	0.0/1.3	0.0/1.3	10.0/1.3	10.0/1.3	18.0/1.3	48.3/1.3	15.0/1.3	25.0/1.3	20.0/1.3
	Str. 25%	35.0/1.3	50.0/1.2	15.0/1.2	15.0/1.2	10.0/1.2	10.0/1.3	17.5/1.2	23.8/1.2	18.5/1.2	43.3/1.3	29.0/1.2	65.0/1.3	15.0/1.2
	Pyr. 10%	95.0/1.3	80.0/1.3	0.0/1.3	50.0/1.3	5.0/1.3	0.0/1.3	18.8/1.3	56.2/1.3	21.0/1.3	45.0/1.3	90.0/1.3	45.0/1.3	25.0/1.3
	Pyr. 25%	100.0/1.3	100.0/1.3	0.0/1.3	65.0/1.3	5.0/1.3	0.0/1.3	36.2/1.3	83.8/1.3	31.0/1.3	50.0/1.3	83.0/1.3	55.0/1.3	25.0/1.3
	Quest	45.0/1.0	5.0/1.0	0.0/1.0	10.0/1.0	0.0/1.0	0.0/1.0	15.0/1.0	3.8/1.0	14.5/1.0	33.3/1.0	20.0/1.0	5.0/1.0	20.0/1.0
	Double	95.0/0.9	75.0/0.9	75.0/0.9	60.0/0.9	15.0/0.9	0.0/0.9	47.5/0.9	53.8/0.9	42.0/0.9	50.0/0.9	70.0/0.9	60.0/0.9	20.0/0.9
	Our τ.01	**100.0/1.2**	**100.0/1.2**	**90.0/1.2**	**95.0/1.1**	**45.0/1.1**	**10.0/1.1**	**71.2/1.1**	**83.8/1.1**	**37.5/1.2**	**51.7/1.2**	**81.0/1.2**	**60.0/1.1**	**25.0/1.1**
	Our τ.1	**100.0/1.4**	**100.0/1.3**	**95.0/1.3**	**90.0/1.3**	**40.0/1.3**	**10.0/1.3**	**70.0/1.3**	**72.5/1.3**	**44.5/1.3**	**56.7/1.4**	**70.0/1.3**	**60.0/1.3**	**30.0/1.3**

Table 8: HELMET per-task quality–latency comparison.

	Method	RAG				LC-QA		ICL					Recall				Rerank.
		KILT-Hotpot	KILT-NQ	KILT-PopQA	KILT-Trivia	InfB-Choice	InfB-QA	ICL-BankI77	ICL-CLINC	ICL-NLU	ICL-TREC-C	ICL-TREC-F	JSON-KV	NIAH-MK2	NIAH-MK3	NIAH-MV	MSMARCO
Llama	Full	60.0/1.0	46.7/1.0	55.0/1.0	80.0/1.0	50.0/1.0	47.1/1.0	90.0/1.0	95.0/1.0	85.0/1.0	80.0/1.0	35.0/1.0	90.0/1.0	90.0/1.0	90.0/1.0	97.5/1.0	15.7/1.0
	Snap 10%	61.7/1.7	44.2/1.6	55.0/1.6	83.3/1.6	50.0/1.6	46.8/1.6	90.0/1.6	95.0/1.6	85.0/1.6	80.0/1.6	35.0/1.6	75.0/1.5	95.0/1.5	35.0/1.5	95.0/1.6	3.3/1.4
	Snap 25%	61.7/1.5	45.0/1.5	55.0/1.4	80.8/1.5	50.0/1.5	48.9/1.5	90.0/1.4	95.0/1.4	85.0/1.4	80.0/1.4	40.0/1.4	90.0/1.4	95.0/1.4	70.0/1.4	98.8/1.5	5.0/1.3
	Str. 10%	46.7/1.7	35.8/1.6	53.3/1.6	78.3/1.6	50.0/1.6	34.8/1.6	90.0/1.5	90.0/1.6	85.0/1.6	90.0/1.5	20.0/1.6	0.0/1.5	25.0/1.5	5.0/1.5	10.0/1.6	2.8/1.4
	Str. 25%	55.0/1.5	37.5/1.5	58.3/1.4	80.8/1.5	50.0/1.5	33.0/1.4	90.0/1.4	95.0/1.4	85.0/1.4	75.0/1.4	20.0/1.4	10.0/1.4	40.0/1.4	20.0/1.4	15.0/1.4	1.4/1.3
	Pyr. 10%	60.0/1.6	45.8/1.6	55.0/1.6	83.3/1.6	50.0/1.6	48.1/1.6	90.0/1.6	95.0/1.6	85.0/1.5	80.0/1.6	25.0/1.6	80.0/1.5	95.0/1.5	45.0/1.5	97.5/1.6	2.1/1.4
	Pyr. 25%	60.0/1.5	44.2/1.5	54.2/1.4	83.3/1.5	50.0/1.4	48.9/1.5	90.0/1.4	95.0/1.4	85.0/1.4	80.0/1.4	30.0/1.4	95.0/1.4	90.0/1.4	80.0/1.4	98.8/1.5	8.5/1.3
	Quest	51.7/0.9	40.8/0.9	47.5/0.9	76.7/0.9	50.0/0.9	47.5/0.9	95.0/0.9	95.0/0.9	80.0/0.9	80.0/0.9	35.0/0.9	5.0/0.9	90.0/0.9	0.0/0.9	93.8/0.9	16.4/0.9
	Double	58.3/1.1	46.7/1.2	55.0/1.1	84.2/1.1	50.0/1.1	47.0/1.1	90.0/1.1	95.0/1.1	85.0/1.1	85.0/1.1	30.0/1.1	25.0/1.0	60.0/1.0	15.0/1.0	57.5/1.1	6.1/1.0
	Our τ.01	**58.3/1.5**	**46.7/1.5**	**56.7/1.4**	**80.0/1.5**	**50.0/1.5**	**47.8/1.4**	**90.0/1.4**	**95.0/1.4**	**85.0/1.4**	**80.0/1.4**	**30.0/1.4**	**90.0/1.4**	**95.0/1.4**	**85.0/1.4**	**97.5/1.5**	**14.0/1.3**
	Our τ.1	**55.0/1.7**	**39.2/1.7**	**55.8/1.6**	**84.2/1.7**	**50.0/1.7**	**45.7/1.7**	**90.0/1.7**	**95.0/1.6**	**90.0/1.6**	**80.0/1.7**	**40.0/1.6**	**90.0/1.6**	**95.0/1.6**	**25.0/1.6**	**53.8/1.7**	**7.5/1.5**
Qwen	Full	41.7/1.0	30.0/1.0	46.7/1.0	77.5/1.0	70.0/1.0	33.5/1.0	85.0/1.0	95.0/1.0	80.0/1.0	95.0/1.0	25.0/1.0	40.0/1.0	50.0/1.0	35.0/1.0	65.0/1.0	6.4/1.0
	Snap 10%	41.7/1.3	29.2/1.3	37.5/1.3	78.3/1.3	70.0/1.3	29.8/1.3	80.0/1.3	90.0/1.3	85.0/1.3	90.0/1.3	25.0/1.3	25.0/1.3	0.0/1.3	0.0/1.3	37.5/1.3	0.8/1.2
	Snap 25%	41.7/1.3	28.3/1.2	46.7/1.2	80.0/1.2	70.0/1.2	28.2/1.2	85.0/1.2	95.0/1.2	75.0/1.2	95.0/1.2	25.0/1.2	30.0/1.2	5.0/1.2	15.0/1.2	60.0/1.2	1.0/1.2
	Str. 10%	40.0/1.3	25.8/1.3	24.2/1.3	62.5/1.3	70.0/1.3	22.7/1.3	80.0/1.3	75.0/1.3	80.0/1.3	85.0/1.3	30.0/1.3	0.0/1.3	20.0/1.3	0.0/1.3	10.0/1.3	1.5/1.2
	Str. 25%	38.3/1.2	22.5/1.3	37.5/1.2	60.8/1.2	70.0/1.2	24.7/1.2	90.0/1.2	90.0/1.2	80.0/1.2	90.0/1.2	25.0/1.2	0.0/1.2	25.0/1.2	10.0/1.2	13.8/1.2	2.2/1.2
	Pyr. 10%	45.0/1.3	26.7/1.3	36.7/1.3	80.8/1.3	70.0/1.3	27.9/1.3	70.0/1.3	95.0/1.3	80.0/1.3	85.0/1.3	25.0/1.3	25.0/1.3	0.0/1.3	0.0/1.3	26.2/1.3	0.0/1.2
	Pyr. 25%	45.0/1.3	26.7/1.3	35.8/1.2	75.0/1.2	70.0/1.2	27.4/1.2	80.0/1.2	90.0/1.2	80.0/1.2	95.0/1.2	30.0/1.2	25.0/1.2	0.0/1.2	0.0/1.2	42.5/1.2	0.0/1.2
	Quest	26.7/1.0	9.2/1.0	14.2/1.0	23.3/1.0	30.0/1.0	7.9/1.0	45.0/1.0	35.0/1.0	10.0/1.0	70.0/1.0	25.0/1.0	0.0/1.0	5.0/1.0	0.0/1.0	31.2/1.0	1.5/1.0
	Double	38.3/0.9	37.5/0.9	37.5/0.9	75.8/0.9	70.0/0.9	30.8/0.9	80.0/0.9	95.0/0.9	75.0/0.9	85.0/0.9	40.0/0.9	0.0/0.9	25.0/0.9	0.0/0.9	33.8/0.9	7.6/0.8
	Our τ.01	**43.3/1.1**	**30.0/1.1**	**46.7/1.1**	**74.2/1.1**	**70.0/1.2**	**31.7/1.1**	**85.0/1.1**	**95.0/1.1**	**80.0/1.1**	**95.0/1.1**	**25.0/1.1**	**45.0/1.2**	**50.0/1.1**	**35.0/1.2**	**65.0/1.2**	**7.4/1.1**
	Our τ.1	**43.3/1.3**	**30.0/1.3**	**46.7/1.3**	**74.2/1.3**	**70.0/1.3**	**32.0/1.3**	**85.0/1.3**	**95.0/1.3**	**75.0/1.3**	**95.0/1.3**	**30.0/1.3**	**45.0/1.3**	**45.0/1.3**	**20.0/1.3**	**57.5/1.3**	**3.5/1.2**